UNIVERSIDADE DE BRASÍLIA
FACULDADE DE TECNOLOGIA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA

METODOLOGIA PARA DESCRIÇÃO DE CÉLULAS ANALÓGICAS COMO IP

JOÃO VITOR BERNARDO PIMENTEL

ORIENTADOR: JOSÉ CAMARGO DA COSTA

DISSERTAÇÃO DE MESTRADO EM ENGENHARIA ELÉTRICA

BRASÍLIA/DF: AGOSTO – 2009

UNIVERSIDADE DE BRASÍLIA

FACULDADE DE TECNOLOGIA

DEPARTAMENTO DE ENGENHARIA ELÉTRICA

METODOLOGIA DE DESCRIÇÃO DE CÉLULAS ANALÓGICAS COMO IP

JOÃO VITOR BERNARDO PIMENTEL

DISSERTAÇÃO SUBMETIDA AO DEPARTAMENTO DE ENGENHARIA ELÉTRICA DA FACULDADE DE TECNOLOGIA DA UNIVERSIDADE DE BRASÍLIA COMO PARTE DOS REQUISÍTOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE.

APROVADA POR:

____

Prof. José Camargo da Costa (ENE-UnB)
(Orientador)

____

Prof. Alexandre Ricardo Soares Romariz (ENE-UnB)
(Examinador Interno)

____

Prof. Vincent Patrick Marie Bourguet (DEE-UFCG)
(Examinador Externo)

BRASÍLIA/DF, 07 DE AGOSTO DE 2009

FICHA CATALOGRÁFICA

PIMENTEL, JOAO VITOR BERNARDO
Metodologia para Descrição de Células Analógicas como IP [Distrito Federal] 2009.

xvii, 174p., 297 mm (ENE/FT/UnB, Mestre, Engenharia Elétrica, 2009). Dissertação de Mestrado – Universidade de Brasília. Faculdade de Tecnologia. Departamento de Engenharia Elétrica.

1.IP	2.VLSI
3.Projeto analógico	4.VHDL-AMS
I. ENE/FT/UnB	II. Título (série)

REFERÊNCIA BIBLIOGRÁFICA

PIMENTEL, J. V. B. (2009). Metodologia para Descrição de Células Analógicas como IP. Dissertação de Mestrado em Engenharia Elétrica, Publicação PPGENE.DM-392/09, Departamento de Engenharia Elétrica, Universidade de Brasília, Brasília, DF, 174p.

CESSÃO DE DIREITOS

AUTOR: João Vitor Bernardo Pimentel

TÍTULO: Metodologia para Descrição de Células Analógicas como IP.

GRAU: Mestre ANO: 2009

____

João Vitor Bernardo Pimentel
SHIN QL 03 conjunto 04 casa 18, Lago Norte
71.505-245 Brasília – DF – Brasil.

AGRADECIMENTOS

Este trabalho não poderia ter sido concluído sem a ajuda de várias pessoas, portanto ficam aqui agradecimentos em especial a:

professor José Camargo da Costa, pela oportunidade e a orientação para desenvolver tudo isso;

professora Janaína Guimarães, pela ajuda nas etapas finais do trabalho;

a equipe do LDCI, particularmente a Genival Araújo, Gilmar Beserra, Heider Marconi, José Edil Guimarães e Vitor Soares;

minha família, pelo apoio;

Sarah Kassim, pela paciência.

RESUMO

METODOLOGIA PARA DESCRIÇÃO DE CÉLULAS ANALÓGICAS COMO IP

Autor: João Vitor Bernardo Pimentel

Orientador: José Camargo da Costa

Programa de Pós-graduação em Engenharia Elétrica

Brasília, agosto de 2009

Este trabalho propõe uma metodologia de descrição de células VLSI analógicas e de sinal misto como blocos de propriedade intelectual (IP). A metodologia foi aplicada em blocos de circuitaria analógica e de sinal misto – um conversor tensão-corrente e um conversor analógico-digital, previamente projetados em tecnologia CMOS – como estudos de caso. Foram realizadas adaptações aos blocos para se adequarem ao contexto de IPs analógicos e construídos modelos de alto-nível dos circuitos, permitindo avaliar sua funcionalidade sem o conhecimento da topologia interna.

Os resultados obtidos dos estudos de caso, principalmente simulações de modelos de alto nível de abstração do circuito, foram analisados para avaliar a metodologia proposta e propôr trabalhos futuros.

ABSTRACT

METHODOLOGY FOR THE DESCRIPTION OF ANALOG CELLS AS IP

Author: João Vitor Bernardo Pimentel

Supervisor: José Camargo da Costa

Programa de Pós-graduação em Engenharia Elétrica

Brasília, august 2009

This work proposes a methodology for the description of analog and mixed-signal VLSI cells as intellectual property (IP) blocks. The methodology was applied on analog/mixed-signal circuitry blocks - a voltage-to-current converter and an analog-to-digital converter, previously designed in CMOS technology - as study cases. Adaptations were performed in the blocks to make them adequate to an analog IP context, and high-level models of the circuits were built, allowing for assessing their functionality with no knowledge of internal architecture.

The achieved results from the study case, especially high abstraction-level simulations, were analysed to evaluate the proposed methodology and to propose future work.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SÍMBOLOS, NOMENCLATURA E ABREVIAÇÕES

A/D	Analógico/Digital
AMS	*Analog/Mixed-Signal*
ASIC	*Application-Specific Integrated Circuit*
BSIM	*Berkeley Short-Channel IGFET Model*
CAD	*Computer-Aided Design*
C_{OX}	Capacitância do óxido de porta
DfT	*Design for Testability*
DNL	*Differential Nonlinearity*
FPAA	*Field-Programmable Analog Array*
HDL	*Hardware Description Language*
INL	*Integral Nonlinearity*
IP	*Intellectual Property*

LDCI	Laboratório de Dispositivos e Circuitos Integrados
K'	parâmetro de transcondutância do transistor
MOS	*Metal-Oxide-Semiconductor*
NMOS	*N-type* MOS
PMOS	*P-type* MOS
RF	Radiofreqüência
SCI	Sistema de Controle de Irrigação
SoC	*Sytstem-on-Chip*
SRS	*Semiconductor Reuse Standard*
VC	*Virtual Component*
V_D	Tensão de dreno
V_{DS}	Tensão dreno-fonte
V_G	Tensão de porta (*gate*)
V_{GS}	Tensão porta-fonte
VHDL	*VHSIC Hardware Description Language*
VHSIC	*Very High Speed Integrated Circuits*
V-I	(Conversor) tensão-corrente
VLSI	Very Large Scale Integration
V_S	Tensão de fonte (*source*)
VSI	*Virtual Socket Interface*
VSIA	*Virtual Socket Interface Alliance*
V_T	Tensão de limiar (*threshold*)

O presente trabalho foi realizado com o apoio do Conselho Nacional
de Desenvolvimento Científico e Tecnológico – CNPq Brasil

1 - INTRODUÇÃO

A evolução da tecnologia para fabricação de circuitos integrados levou, nos últimos anos, a grandes níveis de integração – daí ser chamada de VLSI (do inglês *Very Large Scale Integration*) – , permitindo a inserção de bilhões de transistores em um único chip [1]. Isso motivou o desenvolvimento dos chamados sistemas-em-chip (abreviados por SoC, do inglês *system-on-chip*), em que um chip contém sistemas completos, com seções digitais, analógicas, de RF – todas as funcionalidades necessárias integradas juntas. Embora haja uma tendência para a que maioria das funções implementadas em SoCs sejam implementadas com circuitos digitais ou de processamento digital de sinais, algumas funções (como a interface entre o sistema eletrônico e o "mundo real") sempre permanecerão analógicas [2]. Eventualmente, isso leva à necessidade de serem projetados o que são chamados blocos de sinal misto: blocos que implementam sua funcionalidade tanto através de circuitos analógicos quanto digitais.

O aumento da integração leva também a um aumento da complexidade dos sistemas a serem projetados. Aliado a questões de mercado (especialmente à necessidade de ter-se um produto pronto em tempo menor e especialização do fabricante em setores específicos), isso motivou o desenvolvimento de blocos funcionais para comercialização [3], significando que uma empresa responsável por desenvolver determinado SoC não precisa projetar todo o seu conteúdo, podendo comprar blocos pré-projetados e integrá-los em seu sistema. O responsável pelo SoC poupa tempo no desenvolvimento do produto, e o desenvolvedor do bloco reutilizado pode se especializar, tendo retorno pelo seu projeto sem ter que desenvolver todo o chip ao qual integrá-lo. O conjunto formado por esses blocos funcionais e a documentação associada (como será explicado no capítulo 2) são chamados de IP, do inglês *intellectual property* (propriedade intelectual), porque quem o desenvolveu tem os direitos sobre o seu conteúdo – seu projeto e os circuitos internos do bloco; quem o compra tem o direito de uso, e eventualmente de configurá-lo para seus fins, mas não tem liberdade para modificá-lo à vontade ou de reutilizá-lo em outros projetos (dependendo do acordo inicial). A utilização de blocos de IP é vantajosa também internamente (dentro de uma própria empresa, ou instituição acadêmica, ou qualquer outro desenvolvedor de sistemas microeletrônicos), podendo poupar tempo e diminuir custos no projeto de blocos semelhantes para sistemas diferentes.

Entretanto, a utilização de blocos de IP analógicos ainda é bem menos significativa do que a de blocos de IP digital ([2], [4]). Parte do motivo é que, por circuitos analógicos exigirem otimização mais cuidadosa e projeto dedicado [5], tendem a ser menos reutilizados. Outro problema

é a falta de consenso sobre o que deve ser exigido de um IP analógico [6]. Foi feito um esforço para resolver-se essa questão pela organização chamada *Virtual Socket Interface Alliance* (*VSIA*), que gerou documentos estabelecendo regras e padrões para o desenvolvimento de *componentes virtuais* (*VC*, do inglês *virtual component*), i.e., os blocos de circuito que compõem um IP; porém, a VSIA cessou suas operações [7] e não há, hoje, um padrão amplamente aceito [6].

O objetivo deste trabalho é propor uma metodologia de criação de blocos de IP analógicos e de sinal misto a partir da adaptação de circuitos projetados e otimizados previamente. Serão considerados desenvolvimentos anteriores na tentativa de se padronizar o desenvolvimento de componentes virtuais analógicos, bem como ampla bibliografia sobre o assunto, para chegar-se a uma proposta abrangente. No capítulo 2, serão apresentados aspectos teóricos sobre os quais se baseou o desenvolvimento deste trabalho, inclusive a revisão bibliográfica relacionada. A metodologia elaborada é apresentada no capítulo 3; no capítulo 4, é descrita sua aplicação da metodologia em dois blocos de circuitaria analógica e de sinal misto: um conversor analógico-digital e um conversor tensão-corrente. (Note-se que os projetos dos conversores não fazem parte deste trabalho, tendo sido desenvolvidos previamente no Laboratório de Dispositivos e Circuitos Integrados (LDCI), da Universidade de Brasília.)

Os resultados da aplicação da metodologia proposta nos blocos citados é apresentado no capítulo 5. Uma breve análise do trabalho é feita no capítulo 6.

2 – FUNDAMENTAÇÃO TEÓRICA E DE PESQUISA

2.1 – IP

Como visto no capítulo 1 – Introdução, um IP é, basicamente, um bloco de circuitos implementando determinada função e cujo intuito é o de ser utilizado em sistemas diferentes em que tal função seja seja necessária. Como o usuário não tem acesso direto aos circuitos e outras características internas do IP, este deve ser uma representação conveniente e precisa de um bloco previamente projetado e descrito adequadamente, em termos de modelos e documentação, para que possa ser integrado sem problemas [8]. O que compõe de maneira completa um IP, portanto, não é apenas o circuito (o componente virtual), mas também as informações agregadas a ele. Para circuitos digitais, a descrição de um bloco com determinadas especificações permite com certa

facilidade sua reutilização – por exemplo, a descrição de uma porta lógica ou de um elemento de memória permite que sejam utilizados inúmeras vezes ao longo do projeto, interconectando tantos quantos forem necessários, sem que seja preciso interferir nos circuitos em nível de transistor.

Para circuitos analógicos ou de sinal misto (coletivamente referidos por *AMS*, do inglês *analog/mixed-signal*), a reutilização é mais complicada. Em geral, mesmo quando há algum tipo de síntese de circuito, grande parte do projeto analógico é desenvolvida "manualmente" [6], o que requer tempo e habilidades do(s) projetista(s). O sucesso ou não do circuito analógico, em relação às especificações, depende muito do projeto; os parâmetros do circuito são significativamente definidos pelo leiaute, que podem variar bastante dependendo de sua implementação (embora existam técnicas de leiaute bem estabelecidas [9]). Além disso, blocos de IP analógicos ainda apresentam desafios na verificação de sistemas avançados, devido principalmente às questões de compromisso entre a precisão na caracterização de um bloco e os diversos níveis de abstração em que ele pode ser inserido no projeto.

2.1.1 – Classificação

Em geral, IPs são classificados em *soft*, *firm* ou *hard* [10]. Essa classificação leva em conta o modo como o IP é fornecido: IPs do tipo *soft* são fornecidos como códigos escritos em linguagens de descrição de *hardware* (HDL), através dos quais o usuário pode sintetizar automaticamente um circuito elétrico – em geral, digital [11] (a síntese automática de circuitos analógicos será abordado brevemente no item 2.1.2.2). Isso permite que o conteúdo do VC seja mais acessível ao usuário e também que ele faça alterações que julgue necessárias, resultando em maior flexibilidade do bloco (inclusive quanto à tecnologia de fabricação). O problema de IPs *soft* em relação à proteção de propriedade intelectual é que, embora a topologia dos circuitos internos seja escondida pela abstração do código, é fácil para um usuário fazer modificações no IP e reutilizá-lo sem a permissão ou conhecimento do criador.

IPs do tipo *hard*, por sua vez, são fornecidos como arquivos que definem as máscaras a serem utilizadas na fabricação do circuito integrado; isso significa, por um lado, que o circuito terá menos flexibilidade. Por outro lado, pode ser fornecido já otimizado. Assim, a proteção à propriedade intelectual do criador é maior do que a de IPs *soft* em termos de dificultar a criação de blocos derivados do original. Porém, mesmo que detalhes internos do circuito não sejam visíveis ao

usuário, é possível que a topologia e até o dimensionamento do circuito sejam extraídos de uma análise das máscaras ou do funcionamento do bloco.

IPs *firm* são fornecidos como uma abordagem intermediária, em que o circuito é fornecido como código a ser sintetizado, em que se incluem (por meio de parametrização de determinadas características do circuito) restrições de dimensionamento e posicionamento dos elementos. A utilização desses parâmetros permite uma flexibilidade maior do que em IPs *hard*, porém com desempenho mais previsível do que em IPs *soft*. Circuitos analógicos são fornecidos, geralmente, como IP *hard* (ou *firm,* quando há possibilidade de síntese automática), devido à necessidade de otimização durante seu projeto ([1], [2], [6], [10], [11]).

A descrição em IP de um bloco AMS, portanto, se assemelha muito ao projeto usual do bloco; entretanto, o principal objetivo da descrição em IP é a possibilidade de reutilização, o que traz exigências adicionais. Como, para circuitos analógicos, as especificações de um sistema ou circuito podem ser muito restritivas (devido à sensibilidade a parâmetros mencionada anteriormente), há a necessidade de se poder ter um bloco funcional que possa atender a uma faixa de aplicações e especificações mais abrangente. Isso se torna um problema principalmente em relação à otimização. Por exemplo, um VC que trabalha com sinais de até 50 kHz atende às especificações de determinado sistema que utilize sinais de até 10 kHz. Esta folga, entretanto, pode vir às custas de área física demasiadamente grande, ou de um consumo de potência muito acima do que teria um circuito otimizado para aquelas especificações. É desejável para o usuário, portanto, obter um bloco com especificações as mais próximas possíveis das que precisa.

Há outra classificação, complementar, relacionada à otimização (detalhada em [6]). Circuitos mais próximos aos limites atuais da tecnologia, que incluem decisões mais críticas de velocidade e acurácia, são chamados de *star IP*. Mesmo quando há ferramenta de síntese automática disponível, é indispensável o trabalho manual de projetistas, pelo menos para a parte final de desenho de leiaute do bloco. Por isso, em geral os IPs *star* são fornecidos como IP *hard*: decisões e ajustes necessários à fase de leiaute, que possam comprometer o funcionamento do circuito, já foram tomadas. O circuito em geral não pode ser facilmente reconfigurado (o que não impede sua reutilização, desde que com especificações compatíveis), mas por outro lado teve um projeto otimizado. Os circuitos utilizados como estudo de caso, neste trabalho, seriam classificados como *star IP*.

Quando o bloco inclui circuitos de funcionamento menos crítico ou com especificações menos restritas, é chamado de *commodity IP*. Esses blocos são circuitos mais genéricos, implementando funções mais simples. *Commodity IP*s podem ser fornecidos como IP *soft* sem maiores problemas, pois não há muitas decisões críticas entre a fase de síntese e a finalização. A reconfiguração de tais blocos é muito mais simples, uma vez que o código está disponível e há maior controle sobre síntese e finalização de leiaute. O preço por isso é a menor otimização.

Os circuitos utilizados como estudos de caso, neste trabalho, exigiram otimização cuidadosa e seriam fornecidos como IPs *hard*; porém, implementam funções relativamente simples (como será mostrado no capítulo 4), sem exigências muito críticas de desempenho. Portanto, seriam classificados como *Commodity IP*s.

2.1.2 – Reutilização de IPs analógicos

A bibliografia mostra diferentes abordagens para ampliar as possibilidades de reutilização de um IP analógico. A primeira é a criação de uma biblioteca de variantes do mesmo bloco, cada variante com especificações diferentes. A segunda é possibilitar a reconfigurabilidade do circuito, seja através de parametrização de aspectos do leiaute (para que o usuário defina os parâmetros desejados *antes* da fabricação) ou de estruturas reconfiguráveis (que podem ser alterados *durante* o funcionamento do bloco).

A parametrização do circuito pode, na verdade, ser vista como uma abordagem *de projeto*; é o criador do circuito quem deve ter o cuidado de estabelecer quais parâmetros do projeto impactam em quais especificações. A partir daí, pode disponibilizar versões diferentes do mesmo circuito. Assim, não é o usuário quem deve se preocupar com a parametrização. Basta que ele defina suas exigências para o bloco, e o fornecedor o configura de modo a satisfazê-las [12]. Essa tarefa não é simples, entretanto: há um compromisso entre a complexidade da parametrização e os limites para a otimização do leiaute [13]. Não é necessariamente possível englobar muitas configurações diferentes de leiaute simplesmente através dos parâmetros – até porque deve ser levado em consideração que o processo de fabricação pode alterar ligeiramente aspectos do leiaute.

2.1.2.1 – Biblioteca de células analógicas

Uma possível solução para as dificuldades na reutilização de IPs AMS parece ser o estudo, por parte do criador do IP, do conjunto de configurações alternativas para o bloco. A partir daí, cria-se uma biblioteca de células originadas do mesmo bloco funcional, mas com diferentes especificações (faixas de operação, tecnologia de fabricação, etc.) [5]. O usuário escolhe a versão mais adequada a seu sistema, de preferência com o auxílio de ferramentas de busca e modelos comportamentais ([14], [15]) para avaliar as diferenças entre as configurações sem precisar saber como se deu a parametrização. A célula escolhida, então é fabricada.

A vantagem da adoção de uma biblioteca é que se pode ter todas as células otimizadas. Por outro lado, projetos analógicos comumente são realizados através de uma metodologia *top-down*, em que se parte das especificações funcionais do bloco, escolhe-se uma topologia de circuito, dimensiona-se o circuito, e, por fim, é feito o leiaute (descrição geométrica das máscaras necessárias à fabricação), validando-se cada etapa por simulação antes de seguir-se adiante no fluxo de projeto (como ilustrado na Figura 2.1) [1].
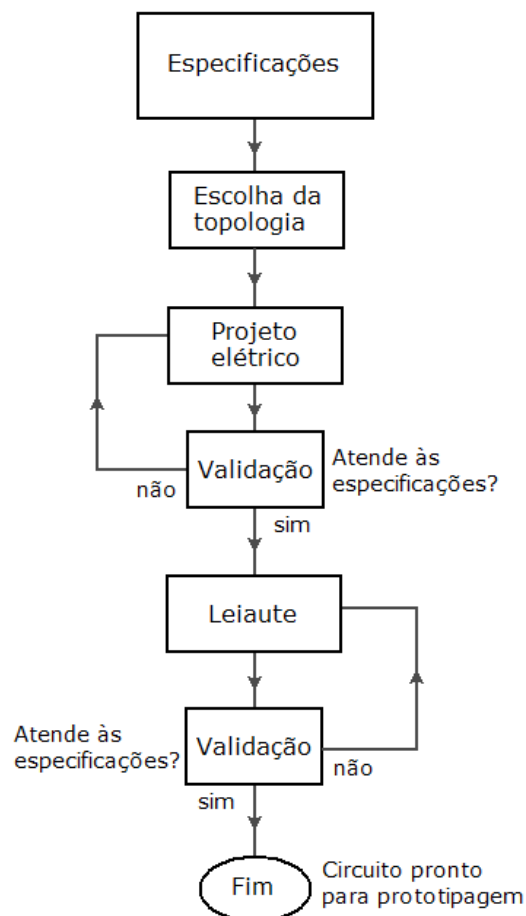
Figura 2.1 – representação de fluxo de projeto analógico

Esse processo tem seu custo, e repeti-lo integralmente para gerar células similares pode ser muito caro, não sendo viável uma biblioteca ampla de células completamente otimizadas. As células em bibliotecas usualmente são variações de um projeto anterior, e por isso talvez não tão otimizadas quanto circuitos de aplicação específica (ASICs, *Application-Specific Integrated Circuits*). Finalmente, os resultados obtidos com síntese automática de circuitos analógicos competem cada vez mais com aqueles obtidos com otimização "manual", conforme os estudos sobre síntese automática de circuitos analógicos se aprofundam ([6], [15]).

2.1.2.2 – Síntese automática de circuitos analógicos

Apesar das dificuldades inerentes à síntese automática de circuitos analógicos, resultados significativos têm sido atingidos [16], abrindo a possibilidade de IPs analógicos *soft* e *firm*. As ferramentas desenvolvidas para síntese automática de blocos AMS utilizam abordagens diferentes – e, freqüentemente, linguagens diferentes (às vezes exclusivas) para a descrição do VC [17] – , mas também há progressos em relação a sintetizar circuitos AMS a partir de HDLs mais comuns, como VHDL-AMS [18], e espera-se que no futuro esse tipo de síntese para criação de IPs se torne mais abrangente e disseminado.

Em geral, a base para a automação da síntese é a parametrização de estruturas analógicas que possam ser reconhecidas, em um código, pela ferramenta de síntese, que constrói um circuito adequado. A maior dificuldade permanece sendo a otimização do leiaute (embora a síntese automática possa mostrar resultados comparáveis ou até melhores do que projeto "manual" [2]), que além de exigir intervenção do integrador (em grau maior ou menor), pode adotar soluções para a síntese física que nem sempre são vantajosas, como a disposição das células em uma malha de linhas e colunas (dificultando a diminuição da área) [6].

2.1.2.3 – FPAA

A possibilidade de se reconfigurar o circuito após a fabricação é ilustrada pelo conceito de *FPAA* (do inglês *Field Programmable Analog Array*), circuitos cuja funcionalidade é controlada através de sinais digitais. Um exemplo é ilustrado na Figura 2.2, onde as chaves modificam as combinações de blocos analógicos do circuito e, portanto, a relação saída/entrada.
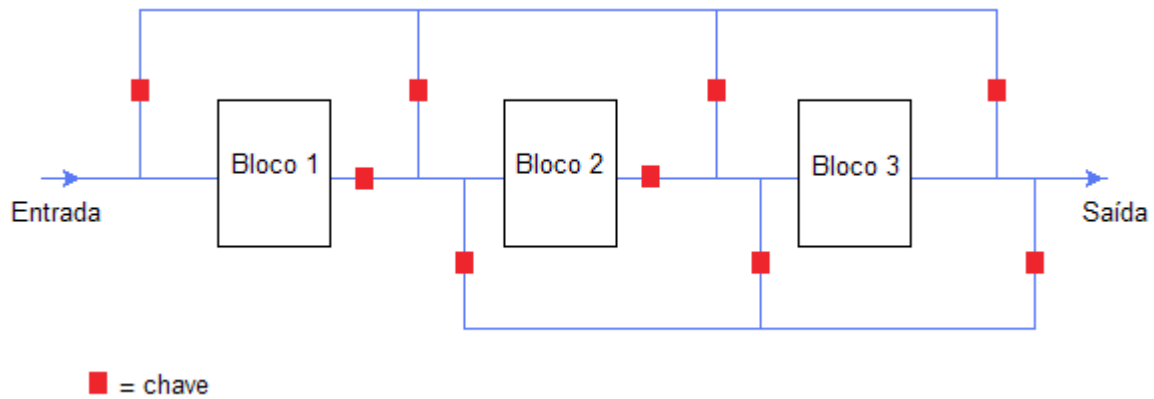
Figura 2.2 – exemplo do conceito de FPAA

2.1.3 – Padronização para IPs analógicos

Não há padronização amplamente aceita para o conteúdo, interface ou comercialização de IPs analógicos. No decorrer deste trabalho, foram estudados os padrões propostos pela VSIA [7] e o *Semiconductor Reuse Standard* (*padrão de reutilização de semicondutor*, abreviado por *SRS*), iniciado pela Motorola [19] e atualmente adotado pela Freescale [20]. Para elaborar o que seria exigido de um IP, a metodologia proposta neste trabalho baseou-se fortemente nos dois padrões citados. Uma breve explicação e análise de cada um é feita a seguir.

2.1.3.1 – Virtual Socket Interface (VSI)

A VSIA era uma organização aberta e internacional composta por representantes da indústria de semicondutores, formada em 1996. Editou uma série de documentos relacionados a circuitos integrados como componentes virtuais (chamados coletivamente de *padrão VSI*), especialmente em relação a critérios que facilitassem reutilização. Encerrou suas atividades em 2008, deixando trabalhos em andamento a cargo de ex-participantes (como o *SPIRIT Consortium* [21], que revisou e mantém o padrão VSIA para identificação de IPs).

De interesse especial deste trabalho é o documento *Analog/Mixed-Signal VSI Extension Specification (AMS 1 2.2)* [22], que trata especificamente de VCs AMS. O restante da

documentação VSI não distingue blocos analógicos de blocos digitais, embora a maior parte de seu conteúdo seja voltada a IPs digitais. O AMS 1 2.2 – cuja última revisão se deu em 2001 – lida com o conteúdo de VCs AMS, formatos de arquivo, recomendações de projeto e integração, etc. Assume que todos VCs analógicos são fornecidos como IP *hard*.

O principal problema do padrão VSIA, em relação à padronização de IPs analógicos hoje, é a sua falta de atualização e o fato de que não atingiu a abrangência a que se propôs.

2.1.3.2 – Semiconductor Reuse Standard (SRS)

O padrão SRS é o resultado formal da compilação de recomendações internas elaboradas, ainda durante a existência da VSIA, pela Motorola [23] – especificamente, na seção responsável por produtos de semicondutores, que mais tarde se tornaria a Freescale [20], que mantém o SRS hoje. Foi considerado superior em certos aspectos [19], e inclusive teve parte de seu conteúdo cedido para a VSIA; porém, embora faça referências a ela, afirma explicitamente que não garante cumprimento de padrões VSI.

Parte da documentação do SRS é abertamente disponibilizado pela Freescale (6 documentos, de um conjunto de 17 [24]; quando se fizer referência ao conteúdo do SRS, neste trabalho, está-se considerando apenas os documentos abertos.) Ainda assim, é essencialmente um padrão interno. Portanto, além de não poder ser adotado em sua totalidade por criadores de IP não associados à Freescale, sua extensa documentação inclui inúmeras exigências e recomendações que se referem a aspectos corporativos dessa empresa, como modelos para documentos (*templates*), *copyright*, etc. – o que o torna por vezes desnecessariamente detalhado para possíveis usuários externos. Não há, como no VSI, documentação específica para IPs AMS, o que também dificulta a adaptação destes ao SRS.

2.2 – MODELAGEM DE CIRCUITOS ANALÓGICOS E DE SINAL MISTO

Para sistemas complexos, modelos de alto nível de abstração são úteis por permitir que diferentes funcionalidades sejam analisadas com relativa facilidade. Simulações elétricas em nível de transistor necessitam de tanto mais esforço computacional quanto maior e mais complexo for o

sistema, porém a funcionalidade pode ser facilmente descrita. Linguagens de modelagem de sistema, como SystemC, e linguagens de descrição de hardware em especial, permitem que o comportamento do sistema seja simulado sem que seja necessário efetuar os cálculos de operação de cada dispositivo no circuito. Existe uma relação de compromisso nessa modelagem: quanto maior o nível de detalhes incorporado pelo modelo, mais complexa será sua interpretação e, assumindo modelos que podem ser simulados, sua simulação exigirá mais esforço computacional [18]. Diferentes níveis de abstração modelam o sistema com mais detalhes (nível de abstração baixo) ou menos (nível de abstração alto), como ilustrado na Figura 2.3.

A modelagem em alto nível é uma abordagem relativamente recente [2] no projeto de circuitos analógicos, tendo grande utilidade tanto nos primeiros estágios de metodologias de projeto *top-down* – quando detalhes da implementação ainda são desconhecidos, impedindo simulação em esquemático – quanto nas etapas finais, de verificação – pois a simulação de sistemas se torna computacionalmente viável com modelos [6].
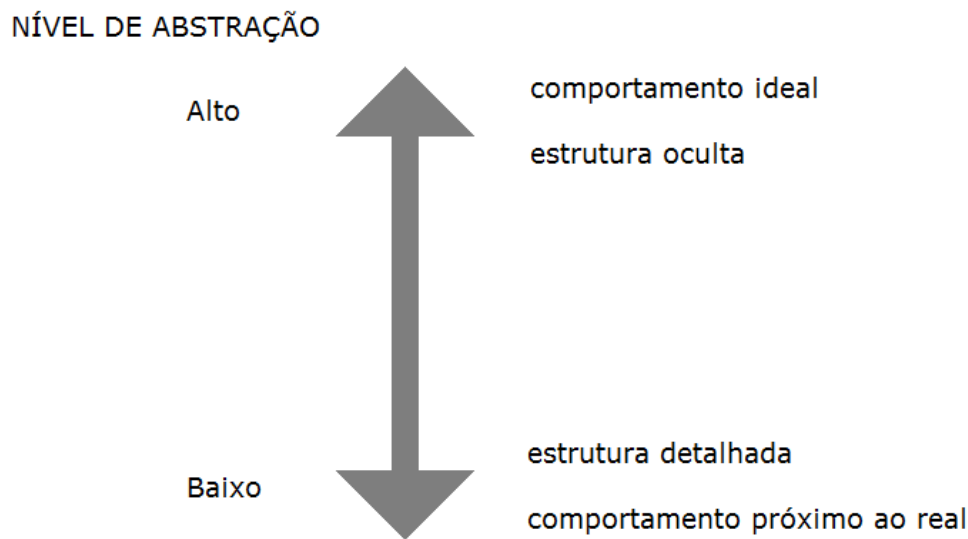
Figura 2.3 – níveis de abstração e detalhamento do sistema

No contexto de criação de IP, as especificações elétricas (condições de operação, faixas de entrada e saída, etc.) geralmente contêm informação suficiente para o comprador avaliar se um determinado bloco é adequado ou não à sua aplicação ou sistema. Entretanto, um modelo do tipo caixa-preta – i.e., que permite observar a interface mas não a construção interna do bloco – que possa ser facilmente simulado é uma ferramenta muito valiosa. Não só integra diferentes especificações do bloco de maneira facilmente verificável, como possibilita que sejam feitas

simulações em conjunto com diferentes sistemas, aplicações e configurações. Assim, compradores podem decidir qual VC, dentre vários disponíveis, é mais útil a seus propósitos. Além disso, modelos de alto nível descrevem o comportamento do circuito sem que sua estrutura interna seja visível; portanto, podem ser fornecidos livremente pelo criador do VC sem que representem ameaça à proteção de propriedade intelectual.

2.2.1 – Níveis hierárquicos de abstração

Para a escolha dos níveis de abstração a serem utilizados na modelagem, foi feito um estudo de trabalhos anteriores. Porém, não há consenso na literatura sobre quais níveis de abstração (ou sob que nomes) devem ser incluídos em um modelo. Diversos exemplos são encontrados nas referências [2], [6], [14] e [25], entre outros. As visualizações para modelos de alto nível dessas referências são as listadas e descritas na Tabela 2.1 (chegou-se aos nomes em português por tradução própria). Neste trabalho, está sendo proposta uma classificação que aponte diferenças significativas sem simplificar demais os modelos. É apresentada na Tabela 2.2.

Note-se que, na proposta da Tabela 2.2, o bloco representado em nível estrutural não necessariamente tem seu comportamento de entrada/saída descrito em mais detalhes do que o modelo comportamental. Entretanto, este nível permite que o comprador ou integrador tenha uma noção melhor sobre o funcionamento do circuito, uma vez que as interações entre sub-blocos é visualizada. A questão a respeito de representações diferentes de um sistema não terem necessariamente nível de detalhamento mais ou menos complexos é abordada em [26] e [27].

Nota-se também que, nas Tabelas 2.1 e 2.2 não se incluem níveis de modelagem elétrica posteriores (esquemático elétrico, leiaute) por se tratarem de visualizações já incorporadas ao fluxo de projeto de circuitos analógicos.

Tabela 2.1 – níveis de abstração encontrados em diversas referências

Referência	Níveis	Descrição
[2]	Funcional	Descrição matemática de comportamento de entrada-saída do bloco.
	Comportamental	Diagrama com sub-blocos considerando comportamento elétrico.
	Macromodelo	Modelo elétrico equivalente, porém com elementos simplificados.
[6]	Comportamental genérico	Modelado antes do dimensionamento do circuito; não considera arquitetura ou aspectos elétricos.
	Comportamental extraído	Modelado após validação do circuito, a partir de características verificadas. Não detalha estrutura.
[14]	Conectivo	Verifica as conexões entre blocos.
	Funcional	Modela equações para implementar a função ideal.
	Comportamental	Modela comportamento elétrico não-ideal, extraído de esquemático.
[25]	Comportamental	Descreve a função ideal do bloco.
	Analítico	Descrição com nível de detalhes equivalente a esquemático elétrico.

Tabela 2.2 – níveis de abstração propostos para modelagem em alto nível

Nível	Conteúdo
Funcional	Comportamento ideal do bloco; não descreve estrutura interna nem sub-blocos.
Comportamental	Incorpora comportamentos não-ideais ao modelo; mantém estrutura oculta
Estrutural	Detalha o comportamento do bloco ao dividi-lo em sub-blocos e modelá-los em descrições comportamentais

2.2.2 – Linguagens de descrição de hardware analógico/sinal misto

Para circuitos digitais, o uso de linguagens de descrição de hardware (HDLs) é amplamente difundido e bem estabelecido. Tais linguagens têm como objetivo possibilitar que circuitos lógicos sejam descritos a partir do seu comportamento, e a partir daí descrevê-los em nível de transistor. Isto é relativamente fácil para circuitos lógicos, em que os estados são bem definidos, mas difícil para circuitos analógicos, que, além de ter uma faixa de valores contínua e ter funcionamento contínuo no tempo, têm seu comportamento fortemente influenciado por parâmetros de tecnologia, topologia, etc.

Existem HDLs dedicadas a modelar circuitos analógicos e de sinal-misto (*HDL-AMS*). Algumas são dedicadas especificamente ao uso de ferramentas de síntese automática, como citado na seção 2.1.2. As mais difundidas atualmente – Verilog-AMS e VHDL-AMS – são, na verdade, extensões de linguagens bem estabelecidas para descrição de hardware digital ([28], [29]). A diferença principal entre HDLs digitais e AMS é que, enquanto as digitais descrevem o funcionamento do sistema em instantes bem definidos, as analógicas permitem uma modelagem contínua. Além disso, há outras diferenças, como permitir modelagem em diferentes domínios de energia [30].

A seguir, será descrita em mais detalhes a linguagem VHDL-AMS, utilizada neste trabalho. Os motivos para sua escolha serão explicados na seção 3.1.5.

2.2.3 – VHDL-AMS

VHDL-AMS é uma linguagem que permite descrever hardware analógico, digital e de sinal misto. É, na verdade, uma conjunto de extensões da linguagem digital VHDL. Foi definida oficialmente em 1999 pelo IEEE, no seu padrão 1076.1 – que foi atualizado diversas vezes, a mais recente em 2007 [29]. A linguagem VHDL-AMS oferece, entre outras capacidades [27], suporte à descrição de: sistemas analógicos em vários níveis de abstração; sistemas conservativos (onde leis de conservação de energia devem ser obedecidas) ou não-conservativos em diversos domínios de energia (elétrico, mecânico, óptico, etc.); sistemas compostos por subsistemas descritos individualmente.

Para o presente trabalho, as principais referências bibliográficas utilizadas na construção de modelos VHDL-AMS foram [27], [29] e [31]. Outras referências utilizadas estão listadas nas Referências Bibliográficas. A seguir, são apresentados alguns fundamentos da linguagem. No Apêndice A são explicados diversos outros conceitos e elementos de sintaxe da linguagem relevantes a este projeto.

2.2.3.1 – Fundamentos

Modelos VHDL-AMS são baseados em *entidades*, que são os blocos de sistema em si. Para cada entidade são definidas suas *portas* – entradas e saídas –, se houver (é válido descrever uma entidade sem portas, como um bloco sem interface externa; as plataformas de teste dos módulos geralmente são entidades sem portas, apenas instanciando as entidades a serem testadas e eventualmente gerando sinais internamente). Assim, a entidade é, fundamentalmente, uma descrição das interfaces do bloco. O seu comportamento é modelado por uma ou mais *arquiteturas*. Cada arquitetura é uma descrição distinta do comportamento da entidade – geralmente, arquiteturas diferentes são utilizadas para níveis de abstração diferentes, mas podem ser utilizadas para descrever funcionalidades diversas de um bloco, ou implementações diferentes de uma mesma função. A parametrização dos modelos é facilitada através das chamadas *portas genéricas*, usadas para definir, através da interface da entidade, parâmetros da arquitetura. Dentro de uma arquitetura, são definidas as *declarações de processo* (*process statements*) e as *declarações simultâneas* (*simultaneous statements*).

Declarações simultâneas são o que permitem à linguagem descrever sistemas analógicos adequadamente. Cada declaração simultânea modela uma equação matemática que atua, a princípio, continua e permanentemente no comportamento do sistema. Na realidade, é possível impor condições para atuação das declarações simultâneas – enquanto as condições forem satisfeitas, o sistema será regido pela declaração durante todo o tempo, e não apenas em instantes pré-determinados, como ocorre em sistemas digitais. Isso permite que, por exemplo, um sistema seja regido por equações diferentes sob situações diferentes, de maneira complementar. Em outras palavras, pode haver descontinuidades no comportamento analógico do bloco. Exemplos simples são um bloco analógico que pode ser ligado ou desligado, ou uma função matemática não-contínua, como exemplificado na Figura 2.4. (O trecho de código VHDL-AMS na Figura pode ser entendido intuitivamente. Detalhes quanto ao uso de códigos desse tipo são apresentados no Apêndice A.)

Casos assim ocorrem nos modelos deste projeto.
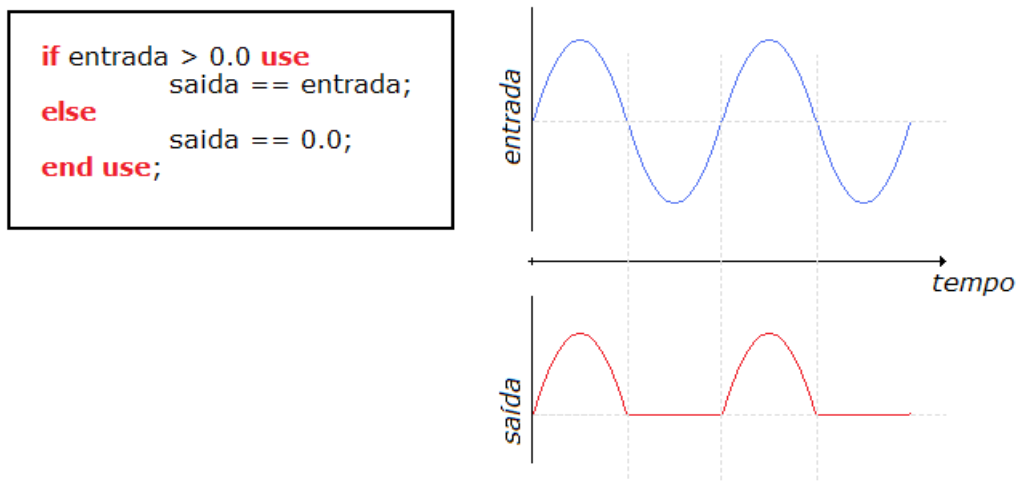
Figura 2.4 – modelagem de funções descontínuas em VHDL-AMS

Declarações de processo (ou *seqüenciais*) são comandos que são executados um após o outro, na ordem em que aparecem no código. Os trechos da arquitetura que contêm as declarações seqüenciais – como atribuição de valores a variáveis ou sinais, execuções condicionais, *loops*, etc. – são chamados de *processos*. São executados do começo ao fim repetidamente (a não ser sob certas condições que suspendem o processo ou que o fazem apenas quando algum estímulo externo ocorre). É importante observar que os comandos em VHDL-AMS não levam nenhum tempo de simulação para serem executados, ou seja, o simulador considera que, em uma seqüência de declarações, todas são executadas sem nenhum intervalo ou atraso, e o resultado é obtido no mesmo instante. Existem, entretanto, maneiras de introduzir uma dependência do tempo em declarações seqüenciais, como através do comando **`wait`**. Esse comando suspende o processo por um tempo definido (p. ex. **`wait for 20 us`** suspende o processo por 20 μs), ou até que determinada condição acontece (p. ex. **`wait until clk = '0'`** suspende o processo até que **`clk`** receba o valor lógico 0), ou, ainda, indefinidamente, utilizando-se somente o comando **`wait`**. Um processo cuja última instrução seja **`wait`** é executado apenas uma vez.

2.2.3.2 – Ferramentas de desenvolvimento

Modelos VHDL-AMS são escritos em forma de texto, podendo ser lidos por ferramentas de projeto adequadas para compilar (interpretar e traduzir os comandos escritos na linguagem em

comandos para a máquina) e simular (interpretar e traduzir os comandos em um conjunto de sinais simultâneos mensuráveis). Não existem ferramentas padrão para a utilização de modelos VHDL-AMS [32], havendo diversas opções comercialmente disponíveis ([33], [34]). Neste projeto, a ferramenta de compilação utilizada foi a NCVHDL [31], e para simulação NCSIM [35], ambas da Cadence Design Systems. Outras ferramentas foram utilizadas para tarefas auxiliares, como a visualização das formas de onda resultantes das simulações.

O uso de *bibliotecas* em VHDL-AMS permite o uso de tipos de sinais, operadores e parâmetros definidos e armazenados previamente, ou padronizados. Em todos os modelos descritos neste trabalho, foi utilizada a biblioteca pré-definida *ieee*, especificamente os pacotes *math_real*, *electrical_systems* e *std_logic_1164*. O comportamento de circuitos elétricos é definido no pacote *electrical_systems*; o pacote *math_real* define operadores e funções matemáticas, como, por exemplo, a constante *pi* (π = 3,14159...), as funções *sin* e *cos*, etc.; o pacote *std_logic_1164* define operações lógicas e determinados tipos de objetos, como vetores de bits.

2.3 – SISTEMA EM CHIP PARA CONTROLE DE IRRIGAÇÃO

O Sistema de Controle de Irrigação (SCI) [36], do qual os blocos utilizados como estudo de caso neste projeto fazem parte, tem como objetivo otimizar o aproveitamento de recursos dentro do que é chamado *agricultura de precisão.* Essa abordagem à produção agrícola leva em conta os prejuízos que irrigação escassa ou demasiada podem causar tanto à lavoura quanto ao meio ambiente, e procura gerenciar a produção com auxílio da medição de diversas variáveis ambientais em diversos pontos da área de cultivo.

No SCI, são espalhadas, pela área de interesse, estações coletoras responsáveis pela caracterização da condição local do solo e pelo acionamento de atuadores para controlar o fluxo de água da irrigação. As estações coletoras (chamadas de *nós*) transmitem e recebem informações de estações de campo por comunicação em radiofreqüência (RF). As estações de campo, por sua vez, comunicam-se com uma estação de base, onde os dados são processados, armazenados, e apresentados ao usuário.

Cada nó é composto por uma bateria, um painel solar (para recarregar a bateria), antena, sensores – de temperatura, pressão mátrica e um monitor da carga da bateria –, atuadores e um SoC,

responsável pelo processamento inicial dos dados obtidos dos sensores, comunicação com as estações de campo e acionamento dos atuadores. Uma representação de um nó do SCI é mostrada na Figura 2.5. São mostradas também as principais seções do SoC. Para o presente trabalho, é de especial interesse a interface analógica.
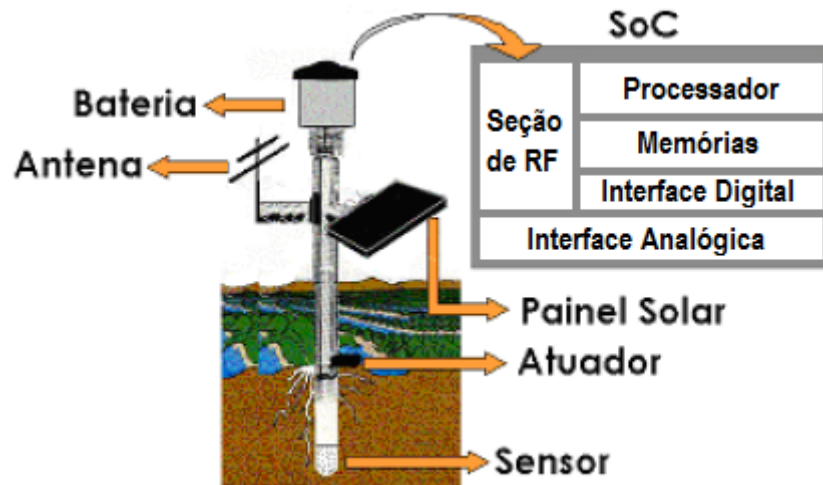
Figura 2.5 – nó do Sistema de Controle de Irrigação

Os sensores ligados ao SoC são dispositivos que capturam informação sobre alguma propriedade física do ambiente e a convertem em sinais elétricos. Entretanto, apenas isso não quer dizer que a informação contida em tais sinais possa ser prontamente utilizada. É necessário um condicionamento anterior de tais sinais para que possam ser fornecidos ao processador do SoC de maneira a serem corretamente interpretados. Os circuitos que realizam este processamento, adquirindo sinais dos sensores e convertendo-os em sinais digitais a serem entregues às seções digitais, compõem a *interface analógica* do SoC, cujo diagrama simplificado é mostrado na Figura 2.6 [37].

Figura 2.6 – interface analógica do SoC do SCI

A conversão dos sinais contínuos no tempo e que podem variar continuamente em determinada faixa – i.e., sinais analógicos – é feita pelo conversor analógico/digital (*A/D*).

Enquanto os sensores provêm sinais de tensão [38], o A/D opera com sinais de corrente; por isso, faz-se necessário um estágio conversor de sinais de tensão para sinais de corrente (*tensão-corrente*, ou *V-I*). Como os diferentes sensores têm características de saída diferentes, o estágio condicionador de sinais é necessário para que a informação seja adequada à entrada do conversor V-I. A seguir, serão apresentados detalhes do conversor A/D e do conversor tensão-corrente.

2.3.1 – Conversor A/D

O conversor analógico/digital do SoC foi projetado para operar com sinais de corrente em topologia do tipo cíclica, com um conjunto de células realizando operações analógicas repetidas sobre um sinal analógico até obter a palavra digital correspondente. Seu funcionamento pode ser resumido nos seguintes passos, ilustrados esquematicamente na Figura 2.7 [39]:

i. O conversor amostra o sinal de corrente em sua entrada, armazenando este valor durante o ciclo de conversão;
ii. O sinal é comparado com uma referência; caso seja igual ou maior, o bit resultante deste ciclo é '1', caso contrário, o bit é '0';
iii. O valor do sinal é multiplicado por 2. Caso o bit obtido no passo anterior tenha sido '1', o valor da referência é subtraído do novo sinal; caso tenha sido '0', o valor da referência é somado ao novo sinal.
iv. O sinal resultante do passo iii é comparado com a referência novamente, obtendo-se o próximo bit.
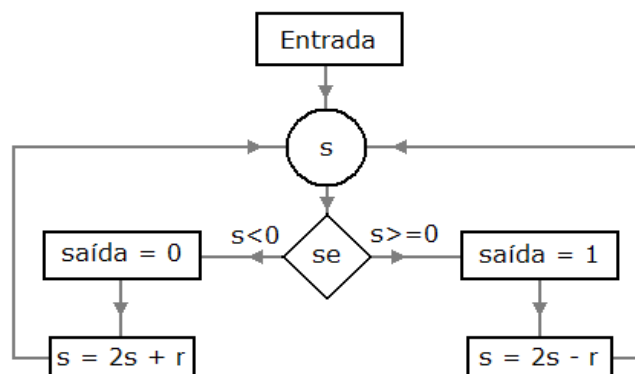
Figura 2.7 – diagrama do funcionamento do conversor A/D

Ao fim de 8 ciclos, portanto, tem-se uma palavra digital de 8 bits representando o valor amostrado inicialmente, sendo que o primeiro bit a ser obtido é o bit mais significativo (MSB, de *most significant bit*). Essa palavra é entregue serialmente na saída binária, também começando pelo MSB. A taxa de amostragem do A/D é de 50 mil amostras por segundo, portanto o processo descrito acima leva 20 μs. Uma série de sinais digitais controlam sua operação, mas não cabe descrevê-los em detalhes aqui e serão abordados quando for relevante. Outras características importantes do A/D são:

- Faixa de entrada: -100 μA a +100 μA
- Referência de comparação: 0 A
- Máxima freqüência do sinal de entrada: 25 kHz
- Opera com sinal de relógio de 16 MHz

Considerando as características acima, vê-se que um LSB (*least significant bit*, bit menos significativo) corresponde a 200 μA/256 = 0,78125 μA, tirando-se daí que o erro de quantização de ±0,5 *LSB* corresponde a ±0,390625 μA. A implementação e operação do conversor A/D são explicadas em detalhes em [39] – [41].

2.3.2 – Conversor V-I

O conversor tensão-corrente do SoC é um bloco puramente analógico. Foi projetado para fornecer, na saída, sinais de corrente adequados à entrada do A/D, a partir de sinais de tensão na entrada. A faixa de saída, portanto, é adequada à faixa de entrada do A/D e ambos os blocos operam na mesma faixa de freqüência. A faixa de entrada do V-I é de 1 V, compatível com pequenos sinais de sensores [38], lembrando ainda que os sinais apresentados à entrada do V-I já terão passado por condicionamento na interface analógica.

A estrutura do conversor tensão-corrente, mostrada na Figura 2.8, é dividida em quatro blocos: um núcleo de conversão, ligado à entrada e já com saída de corrente, uma referência de corrente, um bloco intermediário de ganho, e um bloco de saída.
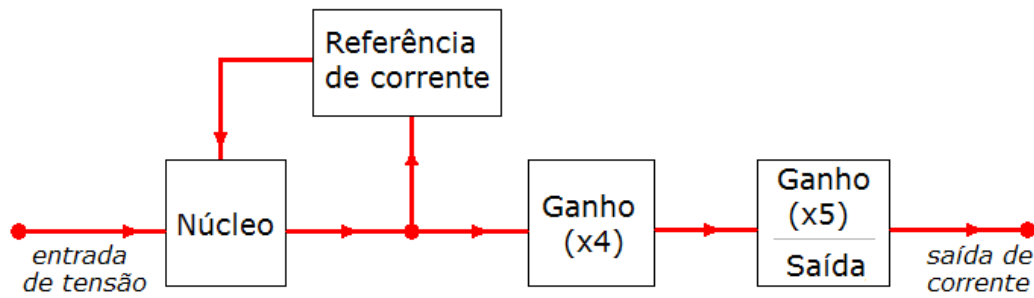
Figura 2.8 – diagrama de blocos do conversor tensão-corrente

O núcleo converte o a tensão de entrada, entre 1 V e 2 V, em um sinal de corrente correspondente, entre 2,5 µA e 12,5 µA. O núcleo requer uma sinal de corrente constante de 1,84 µA, fornecido pela referência. A referência também drena, do nó identificado na figura por *n*, uma corrente de 7,5 µA; assim, o sinal entrando no estágio intermediário de ganho varia entre -5 µA e +5 µA. Esse estágio quadruplica o sinal, fornecendo ao estágio de saída um sinal entre -20 µA e +20 µA. O estágio de saída, por fim, multiplica o sinal por 5, fornecendo na saída do bloco o sinal de corrente na faixa de ±100 µA, conforme a especificação. Detalhes sobre o projeto, validação e implementação do conversor V-I podem ser obtidos em [37] e [42].

3 – METODOLOGIA DE IMPLEMENTAÇÃO DE IPS ANALÓGICOS

3.1 – PROPOSTA

A metodologia de adaptação para IP proposta aqui, se seguida pelo proprietário (criador) de um bloco de circuitos analógicos, tem a intenção de tornar mais fácil a elaboração de um VC a partir de tal bloco, bem como facilitar a reutilização do VC em novos sistemas. Não se tem a pretensão de que seja um padrão a ser seguido por fornecedores ao redor do mundo, mas, justamente pela inexistência de padrões, tentou-se ser abrangente o suficiente para que seja amplamente aplicável.

Esta metodologia, a princípio, pode ser adotada durante o projeto do circuito analógico a ser adaptado, ou depois que pelo menos uma versão do circuito já tenha sido projetada e validada. A idéia, ilustrada na Figura 3.1, é que, uma vez projetado um bloco de circuito analógico, ele pode ser comercializado como IP desde que os seguintes passos sejam tomados:

- Garantir que apresente determinadas características compatíveis com IPs;
- Elaborar modelos de alto nível de abstração;
- Elaborar a documentação adequada.

É claro que, se a criação de um IP for considerada já nas etapas iniciais do projeto do circuito, alguns fatores podem ser levados em conta com antecedência (como a testabilidade), encurtando o tempo necessário com possíveis adaptações futuras e diminuindo a possível diferença de desempenho entre o bloco original e o componente virtual adaptado. Outra consideração é que, como este trabalho não lidou diretamente com síntese automática de células analógicas, a metodologia proposta se aplica apenas a IPs *hard*, conforme descritos na seção 2.1.

Consideradas as condições descritas, o conteúdo deste capitulo descreve características que o fornecedor deve garantir, na adaptação de seu bloco de circuito, para que corresponda ao que se espera de um bloco de IP analógico. São listadas na Tabela 3.1, e detalhadas, uma por uma, em seguida.
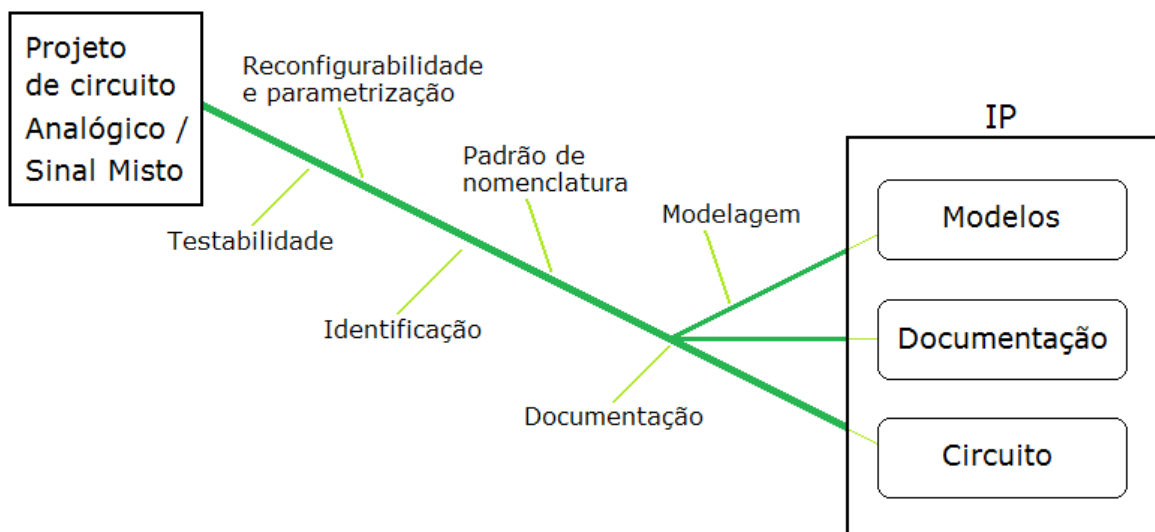
Figura 3.1 – representação da metodologia proposta

Tabela 3.1 – características a serem consideradas na adaptação de um bloco para IP

Característica	Objetivo
Reconfigurabilidade e parametrização	Reuso em diferentes aplicações
Testabilidade	Permitir validação do VC pelo usuário
Padrão de nomenclatura	Facilitar integração e simulação
Identificação	Facilitar integração e proteger propriedade intelectual
Modelagem	Permitir avaliação do VC pelo usuário
Documentação	Permitir a utilização correta do VC

3.1.1 – Reconfigurabilidade e parametrização

A possibilidade de reconfigurar um bloco de circuito para atender a diferentes conjuntos de especificações pode potencializar sua reutilização em diferentes sistemas [4], porém, como discutido no capítulo 2, não deve haver prejuízo de otimização para atingir reconfigurabilidade. Portanto, a metodologia aqui proposta é mais voltada para a criação de uma biblioteca de células analógicas – recomenda-se que o fornecedor invista mais em gerar configurações alternativas do VC, que possam ser escolhidas através da documentação e/ou modelos de alto-nível, do que inserir reconfigurabilidade às custas de desempenho. As possíveis adaptações do bloco a diferentes especificações ou tecnologias podem ser mais vantajosas se abordadas nas etapas de criação e projeto, pelo fornecedor, do que se deixadas a cargo do usuário ([12], [13]).

3.1.2 – Testabilidade

Se um circuito será reutilizado – principalmente como um IP – é importante que o integrador (i.e., quem vai integrá-lo a uma nova aplicação) tenha condições de verificar por si algumas características do circuito, para poder validar seu sistema. Um nível razoável de observabilidade

pode ser conferido ao circuito sem risco à propriedade intelectual de seu criador, e certo nível de controlabilidade é desejável também para que o circuito possa ser validado sob diferentes condições. O usuário deve ser capaz de testar o bloco não só durante sua integração ao sistema, como também após a fabricação, uma vez que esta é efetivamente responsabilidade do integrador [4].

Assim, na criação de um VC, deve-se garantir que haja estruturas de teste no circuito, permitindo observabilidade e controlabilidade (por exemplo, permitindo que tensões em nós internos do circuito sejam medidas, ou que sub-blocos sejam validados separadamente). A elaboração e inclusão de tais estruturas em geral consiste no que é chamado *projeto voltado à testabilidade*, abreviado comumente por *DfT*, do inglês *design for testability*. Existem técnicas de DfT aplicáveis a circuitos analógicos [43]. Caso o circuito projetado previamente não tenha estruturas de testabilidade, elas devem ser projetadas e implementadas de maneira a causar o menor impacto possível na otimização anterior. Se a implementação de testabilidade em um circuito alterar suas especificações (de área, consumo, interface, etc.), isso deve ser documentado apropriadamente.

As especificações VSI ([22], [44] exigem explicitamente que sejam incluídos modos de teste no VC e que sejam fornecidas informações sobre os testes para verificar, após fabricação, a funcionalidade do VC e sua adequação às especificações. Os sinais necessários ao teste (sejam eles digitais ou analógicos) devem ser documentados em detalhes, incluindo parâmetros que descrevam suficientemente os sinais analógicos (limitações de freqüência, amplitude, nível DC...) e digitais (vetores digitais, limitações de velocidade...). O padrão SRS faz exigência semelhante [45], porém apenas em relação a simulação – de que o ambiente de verificação deve ser completamente reproduzível pelo usuário. Também para que a verificação pelo usuário seja a mais próxima possível à realizada pelo fornecedor, recomenda, quanto a testes físicos, que blocos analógicos sejam alimentados isoladamente (o que é prática usual de projeto de circuitos analógicos [1]).

As especificações dos estímulos para teste devem também ser claras em termos da resposta que devem provocar no VC. Por exemplo: não basta dizer que uma tensão *VG*, que pode variar entre 0 e 3 V, controla o ganho do amplificador; é necessário descrever a resposta esperada do VC a tal variação. Caso seja possível prever falhas conhecidas através de resultados característicos dos testes, isso deve ser documentado também.

3.1.3 – Padrão de nomenclatura

Recomenda-se, refletindo abordagens similares da VSIA [46] e SRS [11], que os nomes de pinos, no VC, e principalmente de portas e sinais, nos modelos, sejam dados com todas as letras maiúsculas ou, preferencialmente, todas as letras minúsculas. Além disso, para facilitar a interpretação de simulações, recomenda-se que os nomes de pinos e sinais seja composto de um prefixo de poucas letras indicando o bloco do qual se origina, seguido de um traço inferior e uma expressão adequada à sua função (podendo ser composto de mais de uma parte, separadas por traço inferior). Por exemplo, um pino chamado *ad_d_saida* seria adequado à saída digital de um conversor A/D – indicando o bloco, o tipo de sinal e a função do pino no sistema.

Se for adotado um padrão de prefixos ou nomenclatura, ele deve ser explicado na documentação para que seja corretamente interpretado. Não é importante que sinais e pinos que não são acessíveis ao usuário sigam esse padrão.

3.1.4 – Identificação

O padrão VSI define métodos de rastrear componentes virtuais. Consiste, basicamente, em inserir – como comentários no código de IPs *soft* [47], ou como texto no leiaute de IPs *hard* [48] – informações em forma de texto. Estas informações, chamadas de *etiquetas* (*tags*, em inglês), seguem um padrão que pode ser facilmente localizado. No caso de códigos, o padrão é o descrito abaixo, onde todos os campos são separados por espaços:

<delimitador> <identificador de etiqueta> % *<palavra-chave> <informação>*

O *delimitador* é um conjunto de caracteres que indica que o conteúdo que o segue não é parte do código, e depende da linguagem utilizada – ou seja, é a estrutura utilizada para inserir comentários ("--" em VHDL-AMS, "//" em Verilog-AMS, etc.). O identificador de etiqueta é um conjunto de caracteres padrão, utilizado para facilitar a busca pelas etiquetas no código. No padrão VSI, é "VSIA_Soft_IP_Tag". As palavras-chaves identificam o que o campo seguinte representa. O conjunto "% *<palavra-chave> <informação>* " deve ser repetido tantas vezes quanto forem necessárias, para a inclusão de toda a informação que se deseja incluir na etiqueta. Por exemplo, a etiqueta a seguir, no padrão VSI, identificaria um conversor A/D gerado pelo Laboratório de

Dispositivos e Circuitos Integrados (LDCI) da Universidade de Brasília (UnB), em um código VHDL-AMS:

```
-- VSIA_Soft_IP_Tag % Vendor LDCI-UnB % Product A/D
```

A quantidade de palavras-chave fica a critério do desenvolvedor, porém as utilizadas no exemplo acima – *Vendor* (fornecedor) e *Product* (produto) – são obrigatórias e devem ser os primeiros campos da etiqueta (o padrão VSI traz uma lista de palavras-chave, obrigatórias e opcionais). Os nomes do fornecedor e do produto não precisam seguir nenhum padrão, porém devem ser coerentes entre todos os produtos daquele fornecedor. Outros campos são opcionais e, se utilizados, devem ser documentados. As palavras-chave opcionais (criadas pelo usuário) devem começar com traço inferior e não podem conter espaço em branco, pois a identificação dos campos é feito pela localização do caractere "%" e dos espaços. A informação relativa à palavra-chave, entretanto, pode conter espaços em branco.

Para IPs fornecidos como leiaute, o padrão é semelhante. Deve ser incluído como texto no leiaute, na camada adequada (que não é utilizada na fabricação de máscaras). A estrutura é:

& <palavra-chave> <informação>

A estrutura acima segue as mesmas regras que aquela para etiquetas em IPs *soft*, em relação às palavras-chave obrigatórias e opcionais, uso de espaços em branco, etc. Porém, como em arquivos GDS-II há restrição de tamanho para uma seqüência arbitrária de caracteres, a estrutura deve ser repetida em linhas diferentes, cada uma contendo a informação de determinada palavra-chave (assim, as diferentes palavras-chave serão separadas por quebra de linha e o caractere "&").

Existem maneiras de identificar VCs em um chip sem o uso de etiquetas [49]. No entanto, elas facilitam a integração, principalmente em sistemas complexos utilizando vários componentes virtuais. Por isso, neste trabalho propõe-se a utilização de uma estrutura similar às descritas acima, das quais foi adaptada. Etiquetas em códigos de descrição de hardware (tanto destinados a síntese quanto em modelos) devem ser iniciadas pelo delimitador adequado à linguagem, e ter a seguinte estrutura:

<delimitador> Etiqueta_IP_soft % *<palavra-chave> <informação>*

Etiquetas em IPs *hard* devem ter um identificador de etiqueta também, e serem incluídos no arquivo de leiaute como texto no seguinte formato:

& Etiqueta_IP_hard *<nº de palavras-chave>* & *<palavra-chave>* *<informação>*

O campo inicial visa facilitar a localização das etiquetas, especialmente nos casos em que mais de um leiaute de IP seja utilizado no mesmo projeto. Propõe-se que palavras-chave opcionais sejam da forma "_*<número>*_*<nome>*"; isso, associado ao uso do campo *<nº de palavras-chave>*, garante que, caso sejam necessárias informações em linhas diferentes, elas sejam consideradas como informação relativa ao mesmo IP. O uso de dois campos iniciais também simplifica uma possível varredura automática por etiquetas, já que mantém as informações agrupadas pelo caractere "&" seguido de dois conjuntos de caracteres separados por espaço.

Como campos obrigatórios às etiquetas, propõe-se:

- Fornecedor – identificando o criador do IP
- Produto – identificando o bloco de circuito contido no IP
- Versão – identificando a versão do IP, para não haver ambigüidade

Assim, a etiqueta completa de um código VHDL-AMS seria como abaixo, embora em uma única linha (no presente texto, as quebras de linha visam facilitar a leitura). O til ("~"), em "Versão", foi omitido propositalmente, pois editores de texto frequentemente alteram letras acentuadas, então não é recomendável que a identificação seja feita com palavras acentuadas. Os campos inseridos entre colchetes são opcionais e podem ser repetidos quantas vezes forem desejadas.

```
-- Etiqueta_IP_soft % Fornecedor <informação>
      % Produto <informação>
      % Versao <informação>
     [% <palavra-chave> <informação> ]
```

Para IPs *hard*, a estrutura ficaria como exemplificado a seguir (onde os campos inseridos entre colchetes podem ser repetidos quantas vezes forem desejadas; *n* palavras-chave opcionais resulta em *<número de palavras-chave>* = *n* + *3*).

```
& Etiqueta_IP_hard <número de palavras-chave>
      & Fornecedor <informação>
      & Produto <informação>
      & Versao <informação>
     [& <palavra-chave> <informação> ]
```

3.1.5 – Modelagem

A importância de modelagem de circuitos para a elaboração de IPs já foi discutida no capítulo 2. Propõe-se a criação de modelos simuláveis que representem o VC em três níveis, descritos sucintamente na Tabela 2.2, reproduzida aqui (Tabela 3.1). Os modelos devem ser escritos preferencialmente em HDL-AMS; caso a linguagem utilizada seja específica para determinada plataforma (compiladores/simuladores), é necessário que sejam fornecidas também ferramentas para simulação e/ou modelos equivalentes em linguagens mais amplamente disseminadas. É recomendado – embora não necessário – que sejam escritos modelos equivalentes em mais de uma linguagem.

A linguagem escolhida para escrever os modelos de alto nível foi VHDL-AMS, por diversos motivos. Primeiro, é uma linguagem bem estabelecida, com bibliografia e padronização disponível ([27], [29]). Segundo, permite os níveis de detalhamento desejados neste trabalho. Terceiro, é uma extensão de (e portanto compatível com) VHDL, linguagem na qual foi escrito o controle digital do conversor A/D projetado para o SoC, facilitando inclusive trabalhos futuros envolvendo este bloco. Tal compatibilidade serviu como “fator de desempate” entre Verilog-AMS e VHDL-AMS. E finalmente, as ferramentas utilizadas para projeto de circuitos integrados em nossa instituição suportam o uso de VHDL-AMS.

Tabela 3.2 – níveis de abstração propostos para modelagem em alto nível

Nível	Conteúdo
Funcional	Comportamento ideal do bloco; não descreve estrutura interna nem sub-blocos.
Comportamental	Incorpora comportamentos não-ideais ao modelo; mantém estrutura oculta
Estrutural	Detalha o comportamento do bloco ao dividi-lo em sub-blocos e modelá-los em descrições comportamentais

Embora já exista uma descrição comportamental das seções digitais do SoC em SystemC, a modelagem de circuitos analógicos com SystemC-AMS, um conjunto de extensões para SystemC, foi descartada para o momento. O principal motivo para isso é que as extensões AMS para SystemC são ainda muito recentes – a primeira versão do manual de referência [50] foi publicada em dezembro de 2008. Espera-se que tais extensões ainda passem por uma fase de amadurecimento antes de estabilizarem-se. Além disso, é vantajoso explorar a possibilidade de co-simulação entre diferentes linguagens de descrição de sistema ou hardware para que a metodologia adotada seja aplicável com menos restrições. (Infelizmente, quanto a este último ponto, houve certa frustração no curso do trabalho, pois a documentação das ferramentas [51] afirmava a possibilidade de co-simulação de blocos VHDL-AMS e SystemC. Porém, após diversas tentativas e contato com o serviço de suporte, julgou-se que as ferramentas ainda não estão preparadas para que simulem blocos SystemC instanciando blocos analógicos VHDL-AMS. Uma descrição deste erro deve ser incluída em atualizações futuras das ferramentas.)

3.1.5.1 – Interface

Os modelos devem descrever a funcionalidade do bloco e sua interface; porém, é aceitável que, para simplificar os modelos e sua simulação, sejam omitidos alguns pinos de entrada e saída. Como o objetivo dos modelos é facilitar a compreensão do bloco e permitir simulações mais rápidas, é um certo contra-senso exigir que os modelos apresentem interface idêntica à do VC, ainda que parte da interface não seja relevante a simulações de alto-nível. Para manter a coerência entre os modelos e o bloco real, os pinos de entrada e saída que forem modelados devem ter nomes compatíveis com aqueles no leiaute. É recomendado que os modelos considerem todas as restrições de nomenclatura para HDL analógico conforme descritas no Apêndice B – e não apenas aquelas específicas à linguagem em que estão sendo escritos – para que possam ser facilmente adaptados para outras linguagens ou inseridos em co-simulações. (Embora o Apêndice B considere apenas VHDL-AMS e Verilog-AMS, pode vir a ser expandido no futuro.)

3.1.5.2 – Representação geométrica

A interface completa do VC deve ser descrita – inclusive em termos de formato e área, para permitir *floorplanning* (descrição gráfica da geometria do circuito, com delimitação da área de cada

sub-circuito) – mas isso não requer um modelo simulável. Uma descrição meramente gráfica da geometria externa e posicionamento de pinos do VC é facilmente elaborada pelo fornecedor e interpretada pelo integrador, não sendo necessária a elaboração de um modelo em HDL (como recomendado pelo padrão VSI [46]).

3.1.5.3 – Representação da topologia interna

Os sinais presentes no modelo devem representar, principalmente, a funcionalidade do bloco. A modelagem da estrutura interna pode ser mais ou menos detalhada, dependendo da importância de sua descrição para o entendimento e análise do bloco. Mesmo que a estrutura interna seja representada, os sinais internos, no modelo, não precisam ser fiéis àqueles no circuito real – pois, se os sinais não serão observáveis ao usuário do IP, não há, de fato, necessidade de uma representação acurada no modelo. Desde que os sinais nos limites da interface representem os sinais reais, os sinais internos podem ser tão abstratos quanto se queira. O SRS recomenda, inclusive, que os modelos não permitam que sinais internos sejam "forçados" [45] (i.e., que se dêem valores arbitrários a sinais que não podem ser acessados diretamente no VC), e que, embora possam ser visualizados em simulação, o controle se dê apenas a partir dos limites do VC visíveis ao usuário.

3.1.6 – Documentação

As especificações apresentadas nesta seção para a documentação entregue como parte de um IP baseiam-se fortemente nos padrões VSI e SRS, especialmente nos documentos [22] e [11]. O conteúdo descrito nesses padrões foi estudado e comparado para elaborar a proposta de documentação que se segue.

As informações relevantes ao bloco devem ser divididas de forma organizada, em documentos separados com conteúdos bem definidos. Ambos os padrões citados concordam sobre isso, embora adotem organizações bastante distintas do conteúdo. Devem ser incluídas descrições textuais e gráficas do VC – funcionalidade, diagramas de bloco, informações de desempenho, notas de aplicação (i.e., exemplificando o comportamento do VC em uma aplicação típica), detalhes sobre a interface, tanto analógica quanto digital (se houver), princípios de funcionamento, entre outros tópicos. É recomendado que toda a documentação siga às seguintes orientações:

- Usar unidades do Sistema Internacional [52]; caso sejam utilizadas unidades diferentes, identificá-las e compará-las a grandezas do SI.
- Ao escrever datas, expressar o dia e ano numericamente, e o mês pelo nome (podendo ser abreviado), para não haver ambigüidade (p.ex.: “07 de Julho, 2009”, “10 FEV 2009”);
- Utilizar legendas para figuras e tabelas;
- Identificar versões dos documentos por “x.y”, onde *x* indica alterações significativas (devido a alterações no bloco, por exemplo) e *y* indica alterações menores (correção de informações no documento, por exemplo).
- Reduzir redundância na documentação referindo-se a outros documentos quando necessário; caso sejam feitas referências a documentos que não fazem parte da documentação do IP, incluir uma lista de Referências Bibliográficas.
- Cada documento tem certas seções obrigatórias. Caso não haja informação disponível para determinada seção, ou a seção não seja aplicável ao VC, indicá-lo sem subtrair a seção do documento;
- A numeração das seções dos documentos deve seguir a ordem “x.y”, onde *x* é o número do documento e *y* o número da seção. Subseções podem ser numeradas com quantos subníveis forem necessários (“x.y.z.t...”). Se forem incluídas seções opcionais, numerá-las mantendo a coerência com a numeração anterior do documento, para evitar que referências de um documento a outro sejam invalidadas.

É apresentada a seguir (Tabela 3.3) a divisão proposta, com uma breve explicação do conteúdo de cada item. Tais conteúdos já foram abordados ao longo deste trabalho, portanto não se considerou necessário aqui aprofundar-se mais em seu significado. Com exceção do Resumo, todos os documentos listados na Tabela 3.3 devem incluir as seguintes informações:

- Folha de rosto identificando:
    - Fornecedor
    - Nome e versão do VC
    - Nome e versão do documento
- Histórico de versões do documento
- Introdução (item número 1 em todos documentos)
- Lista de Figuras e Tabelas (se houver)
- Referências Bibliográficas (se houver)
- Número de páginas

Tabela 3.3 – documentação proposta

Documento	Descrição	Conteúdo
Resumo	Descrição curta do VC, em uma única página, incluindo apenas informações essenciais	Funcionalidade do bloco
		Principais especificações
Guia do Usuário	Descrição detalhada das especificações e funcionalidade do VC	Especificações detalhadas – faixas de operação (entrada, saída, temperatura, freqüência)
		Descrição detalhada da funcionalidade
		Descrição da estrutura (diagrama de blocos)
		Sinais externos
		Modos de operação
		Problemas conhecidos
		Exemplo de aplicação
Guia de Criação	Informações sobre a criação do VC	Histórico de versões do bloco
		Histórico do VC (motivação, origem)
		Princípios de funcionamento
		Metodologia utilizada no projeto
Guia de Teste	Informações sobre teste e validação	Detalhamento das estruturas de teste
		Descrição detalhada de modos de teste e resultados esperados
		Embasamento para suposições (condições para que a validação do VC seja aplicável)
Implementação Física	Informações sobre a implementação física do VC	Representação geométrica (para *floorplanning*)
		Localização dos pinos
		Tecnologia(s) de fabricação utilizada(s)
		Especificações de *pads*
Modelos	Modelos de alto nível e suas descrições	Códigos de modelos de alto nível de abstração
		Descrição da hierarquia utilizada nos modelos
		Requisitos para simulação e validação
		Códigos de plataformas de teste (*testbenches*)
		Descrição detalhada do funcionamento dos modelos
		Descrição da parametrização utilizada

O *Resumo* tem o intuito de ser uma consulta rápida, não devendo incluir folha de rosto ou lista de figuras e tabelas. O *Guia do Usuário* contém informações sobre o VC – o funcionamento do circuito, as especificações elétricas e ambientais, modos de operação do VC, enfim: informações essenciais à utilização do bloco. O *Guia de Criação* deve informar o usuário sobre a origem do VC – se foi projetado originalmente para determinada aplicação; se partiu de projeto anterior; as versões anteriores do VC; seus princípios de funcionamento (p.ex., "o funcionamento do conversor A/D é baseado no uso de células copiadoras de corrente...") etc. O *Guia de Teste* deve conter as informações necessárias para a validação do VC, inclusive descrevendo as condições de teste para as quais a validação apresentada é garantida (condições de temperatura, de alimentação elétrica, plataformas de teste, etc.). O documento chamado de *Implementação Física* deve dar informações para que o usuário integre o VC ao leiaute de outros blocos, como localização e tipos de *pads* (digital, RF,...), e processo de fabricação. Por fim, o documento chamado de *Modelos* fornece os códigos para os modelos de alto nível e informações relacionadas a eles, como a linguagem utilizada (indicando referências), como a parametrização do modelo se relaciona às implementações reais do bloco – enfim, informações suficientes para que o usuário, se desejar, seja capaz de criar um próprio modelo equivalente ao fornecido.

Na Tabela 3.4, a divisão proposta é comparada com a documentação exigida pelos padrões VSI e SRS. A construção da tabela parte do conteúdo dos documentos na presente proposta, indicando em quais documentos das referências ele é majoritariamente abordado.

Tabela 3.4 – comparação entre documentação proposta e referências

Proposta	VSIA	SRS
Resumo	(não há documento correspondente)	6.5 IP Brief
Guia do Usuário	2.1 User Guide 2.3 System Architecture and Design	6.6 Creation Guide 6.7 Block Guide
Guia de Criação	2.1 User Guide	6.6 Creation Guide
Guia de Teste	2.1 User Guide 2.5 Test Support	6.10 Test Guide 6.11 Verification Guide
Implementação Física	2.1 User Guide 2.2 Process Definition 2.6 Physical Block Implementation	6.9 Integration Guide
Modelos	2.3 System Architecture and Design 2.4 Functional and Performance Modeling	6.11 Verification Guide

Como o conteúdo de determinado documento em um padrão não corresponde exatamente ao conteúdo de um documento em outro padrão (pode ser dividido em vários documentos), há referências que aparecem mais de uma vez na Tabela 3.4. Os números se referem à numeração utilizada nas referências [11] e [22] para descrever cada documento.

4 – APLICAÇÃO DA METODOLOGIA (ESTUDOS DE CASO)

A metodologia proposta foi aplicada ao conversor tensão-corrente e ao conversor analógico/ digital do SoC do SCI. Sua implementação está descrita no restante deste capítulo, enquanto os resultados serão apresentados e discutidos no capítulo seguinte. Como os blocos utilizados como estudos de caso já satisfaziam parcialmente a algumas das exigências e recomendações propostas e já havia documentação parcialmente elaborada (embora não no padrão proposto), o foco principal, neste trabalho, foi em relação à modelagem dos circuitos.

4.1 – CONVERSOR V-I

A adaptação do conversor V-I à metodologia proposta abordou modelagem em alto nível (em VHDL-AMS) e documentação, mas além disso foram feitas também algumas alterações ao circuito previamente projetado, com o objetivo deixar o bloco mais adequado ao contexto de IPs analógicos.

4.1.1 – Adaptação do bloco

Os circuitos que compõem o conversor tensão-corrente, como mostrado na seção 2.3.2, são puramente analógicos. A sua utilização como IP analógico encontra um problema: para que seja utilizado sem que sua estrutura interna seja visível, o conversor apresentaria apenas um pino de entrada e um pino de saída. Por mais que isso seja suficiente para implementar sua funcionalidade, é desejável que haja algum nível de observabilidade e controlabilidade no circuito [2]. Somente a adição de pinos para medição de tensões de nós internos durante o funcionamento não seria suficiente (até porque o conversor trabalha com sinais de corrente). Neste trabalho, é proposta a adição de chaves e pinos de saída ao circuito do conversor (cuja representação é repetida, para fins

de comparação, na Figura 4.1(a)), chegando-se à solução ilustrada na Figura 4.1(b).
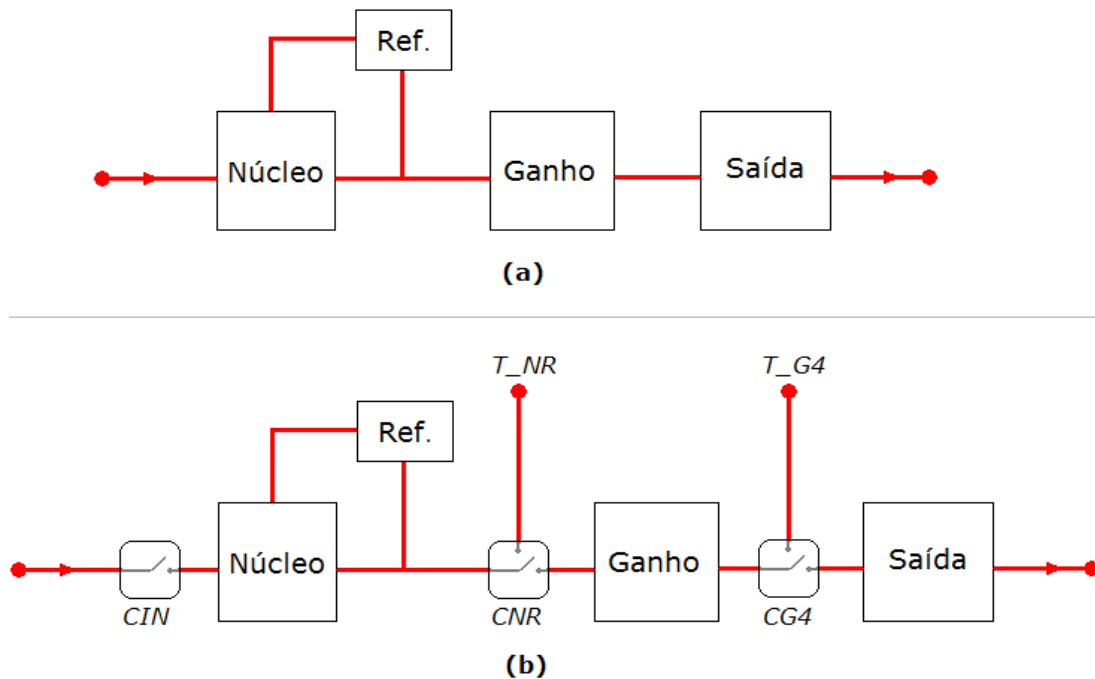
Figura 4.1 – diagrama do conversor V-I (a) original; e (b) com chaves para testabilidade

O controle das chaves permite que o fluxo dos sinais seja interrompido antes do núcleo de conversão (através da chave CIN, na figura), antes do estágio intermediário (através da chave CNR) ou antes do estágio de saída (através da chave CG4). Além disso, permite que o sinal de corrente seja desviado de pontos intermediários do circuito para os pinos de saída T_NF e T_G4, possibilitando portanto verificar o funcionamento dos sub-blocos do circuito. Portanto, o que se implementou foi, efetivamente, a adição de um "modo de teste" na operação do conversor V-I. Os nomes dados às chaves e pinos refletem essa funcionalidade: as chaves e sinais de controle são nomeadas por *C*, de *controle*, e os pinos por *T*, de *teste*. As duas letras seguintes representam o sub-bloco que cada chave (associada a um pino de saída, exceto no caso de CIN) permitem observar: assim, *T_NR* é o pino que, em modo de teste, permite medir a corrente resultante da combinação do núcleo e da refererência de corrente; *T_G4* permite observar, em modo de teste, a corrente vinda da saída do bloco de ganho igual a 4. Note-se que a verificação da funcionalidade do núcleo isoladamente não foi implementada pois este sub-bloco só funciona em conjunto com a referência de corrente; e não foi implementada observabilidade extra para o bloco de saída, pois uma vez caracterizadas as funcionalidades dos estágios anteriores, a medida da corrente de saída é suficiente para verificar o estágio de saída.

A representação na Figura 4.1 é simplificada; as chaves CNF e CG4 são, na realidade, um par de chaves complementares, conforme será visto detalhadamente adiante; mas essa representação simplificada auxilia a compreensão dos modos de operação do conversor. Na operação normal do conversor, os sinais de controle das três chaves devem estar em nível alto (tensão de 3,3 Volts, correspondente à tensão de alimentação do bloco). Aplicar tensão de 0 V (ou seja, nível baixo) em alguma das chaves leva às seguintes situações:

- CIN em nível baixo: a entrada do núcleo de conversão é cortada, portanto a saída do bloco será nula independentemente do sinal de entrada;
- CIN em nível alto, CNR em nível baixo: a entrada do estágio intermediário de ganho é cortada, e o sinal de corrente é desviado para o pino de saída T_NR; esta configuração permite observar a corrente resultante da combinação entre núcleo e referência;
- CIN e CNR em nível alto: a entrada do estágio de saída é cortada, desviando o sinal de corrente oriundo da saída do estágio intermediário para o pino T_G4.

As situações acima estão resumidas na Tabela 4.1 (onde "1" representa nível alto e "0", nível baixo). Nem todas as combinações estão representadas; as combinações restantes, desnecessárias, serão abordadas no Capítulo 5.

Tabela 4.1 – modos de operação do conversor V-I alterado

CIN	**CNR**	**CG4**	**Operação**	**Resultado**
0	**0**	**0**	Desligamento	Saída do bloco nula
1	**0**	**1**	Teste (NR)	Saída no pino T_NR, na faixa de ±5,0 μA
1	**1**	**0**	Teste (G4)	Saída no pino T_G4, na faixa de ±20,0 μA
1	**1**	**1**	Normal	Saída do bloco na faixa de ±100,0 μA

A testabilidade do conversor V-I foi implementada utilizando-se chaves MOS simples. O objetivo não é ter-se propriamente um "circuito chaveado", onde a velocidade e sincronia de chaveamento é crucial, como é o caso do conversor A/D, mas simplesmente ter controle de modos de operação diferentes. Por isso, a utilização de chaves mais complexas (com compensação de injeção de cargas, etc.) foi descartada para priorizar a simplicidade do circuito e manter o baixo custo e baixo consumo.

Na Figura 4.2, mostra-se mais detalhadamente a implementação das chaves utilizando transistores NMOS e PMOS. Uma tensão nula no pino *vi_d_cin* leva o transistor *CIN* ao corte; isso

não leva a tensão na entrada do núcleo a 0 V, mas a um valor abaixo da faixa de entrada que resulta em corrente nula em sua saída. O par *CNRn/CNRp* compõe o que está sendo chamado de "chave *CNR*", e o par *CG4n/CG4p* compõe a chave *CG4*. Quando a tensão no pino *vi_d_cnr* for alta (3,3 V), *CNRn* estará conduzindo e *CNRp* estará em corte, de forma que (idealmente) toda a corrente é transmitida para o estágio intermediário; quando a tensão for baixa (0 V), *CNRp* estará conduzindo e *CNRn* estará em corte, de forma que a corrente será desviada para o pino *vi_t_nr.* O par *CG4n/CG4p* opera de forma análoga. Assim, se implementa o modo de teste descrito.
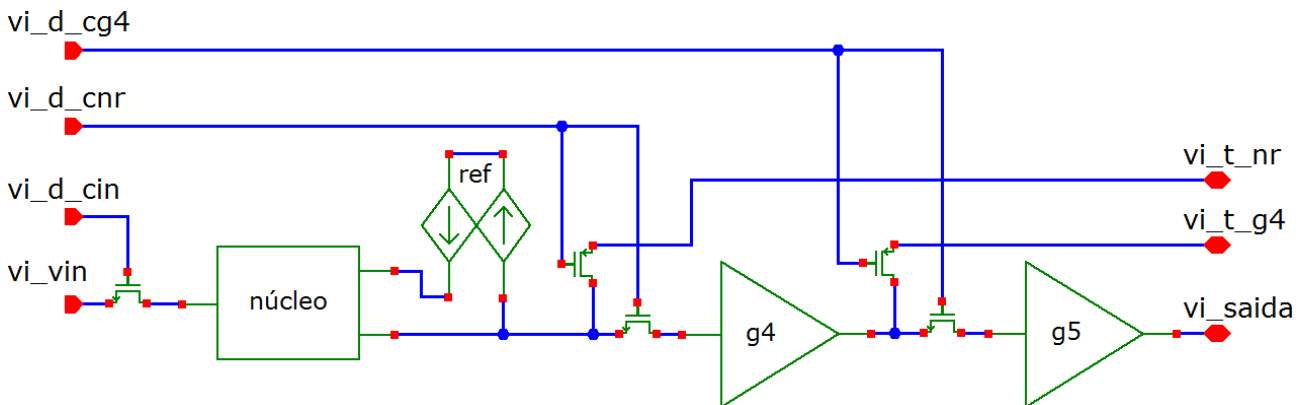
Figura 4.2 – representação esquemática do conversor V-I modificado

Uma preocupação no uso das chaves era a relação de compromisso entre baixa resistência e tamanho pequeno do transistor, que surge devido ao fato de que, quanto mais largo o canal do transistor, menor a resistência. As chaves N foram dimensionadas com W = 10 µm e L = 1 µm, dessa forma mantendo uma resistência relativamente baixa (em torno de 700 Ω) com dimensões razoáveis. A Equação (4.1), abaixo, expressa a resistência de chaves NMOS relacionada a alguns parâmetros do transistor [54], quando este está sob certas condições (todas satisfeitas pelas chaves em condução no conversor V-I, verificadas nas simulações do circuito).

$$R_{ON} = \frac{1}{K' \cdot (\frac{W}{L}) \cdot (V_{GS} - V_T)} \tag{4.1}$$

para:

- $V_{GS} > V_T$;
- $V_{DS} << 2(V_{GS} - V_T)$;
- $V_G = V_{DD}$

Na terminologia utilizada, V_G é a tensão na porta (*gate*) do transistor, V_S é a tensão na fonte

(*source*) e V_D é a tensão no dreno; V_T é a tensão de limiar; V_{ab} indica a tensão entre dois terminais, onde *a* e *b* podem ser porta, fonte ou dreno; *W* e *L* são a largura e o comprimento do canal, respectivamente, e *K'* é o parâmetro de transcondutância, igual ao produto da mobildade de superfície do canal (μ_0) e a capacitância do óxido de porta (C_{OX}).

As chaves P, a princípio, seriam dimensionadas para ter a mesma resistência das chaves N, para que os sinais no modo de teste correspondessem mais fielmente àqueles da operação normal do conversor. Entretanto, há dois problemas nessa abordagem: primeiro, a resistência do transistor em condução, R_{ON}, não é constante, mas varia com a tensão (e corrente) de entrada, e como os parâmetros de processo de transistores N e P são diferentes, suas resistências podem ser igualadas para um determinado valor de tensão e corrente na operação, mas não serão iguais para toda a faixa – portanto, distorções são inevitáveis. Em segundo lugar, devido à diferença nos parâmetros *K'* dos transistores PMOS e NMOS na tecnologia utilizada no projeto e fabricação do conversor V-I [54], as chaves P teriam tamanho até cinco vezes o das chaves N.

Pelos motivos apresentados, então, optou-se por manter as chaves P com o mesmo tamanho das chaves N (resultando em resistência em torno de 3 kΩ), o que tornou possível manter o tamanho do circuito dentro dos limites estabelecidos pelo seu projeto anterior [37]. Projetado o modo de teste, o desempenho do circuito alterado foi comparado com o circuito original. Verificada a compatibilidade (os resultados são discutidos no capítulo 5), foi desenhado o leiaute do novo circuito tomando como base leiautes prévios dos sub-blocos.

4.1.2 – Modelagem em alto nível

A seguir será explicado como foram construídos os modelos VHDL-AMS do conversor V-I. Os resultados serão apresentados no capítulo 5, e os códigos completos no Apêndice E.

4.1.2.1 – Modelo funcional ideal

A arquitetura *vi_ideal* do conversor V-I implementa sua funcionalidade ideal em um bloco único, com entrada de tensão e saída de corrente. Sua declaração de entidade inclui as seguintes portas (relacionadas àquelas descritas na seção anterior):

- vi_vin terminal elétrico
- vi_saida terminal elétrico
- vi_t_nr terminal elétrico
- vi_t_g4 terminal elétrico
- vi_d_cin sinal binário
- vi_d_cnr sinal binário
- vi_d_cg4 sinal binário

A relação de conversão é dada pela equação (4.2), onde *v* é a entrada de tensão e *i*, a saída de corrente:

$$i = (v - 1.5)\times 2.0\times 10^{-04} \tag{4.2}$$

Assim, valores de *v* entre 1.0 e 2.0 correspondem a valores de *i* entre -100×10^{-06} e $+100\times 10^{-06}$. Essa relação é traduzida no código por uma declaração simultânea:

```
vi_i_saida == (vi_v_in-1.5)*2.0e-4;
```

Pode-se notar que não foi necessário transformar os tipos das quantidades `vi_i_saida` (corrente associada ao terminal de saída) e `vi_v_in` (tensão no terminal de entrada). Ambos os tipos (*current* e *voltage*, respectivamente) são subtipos definidos, no pacote *ieee.electrical_systems*, a partir da natureza *electrical*, de forma que seus valores se relacionem diretamente, desde que expressos como números reais.

A opção de zerar a saída independentemente da entrada é implementada inserindo a declaração simultânea acima em uma estrutura `IF` simultânea e instruindo o simulador que há uma descontinuidade associada ao sinal de controle `vi_d_cin`:

```
IF vi_d_cin = '1' USE
     vi_i_saida == (vi_v_in-1.5)*2.0e-4;
ELSE
     vi_i_saida == 0.0;
END USE;

break on vi_d_cin;
```

A estrutura acima é uma simplificação para ilustrar o uso do sinal de controle vi_d_cin; com

todas as chaves e pinos implementados, o código fica da seguinte forma:

```
IF vi_d_cin = '1' USE

      IF vi_d_cnr = '1' and vi_d_cg4 = '1' USE
            vi_i_saida == (vi_v_in-1.5)*2.0e-4;
            vi_i_tnr == 0.0;
            vi_i_tg4 == 0.0;

      ELSIF vi_d_cnr = '0' USE
            vi_i_saida == 0.0;
            vi_i_tnr == (vi_v_in-1.5)*1.0e-5;
            vi_i_tg4 == 0.0;

      ELSE
            vi_i_saida == 0.0;
            vi_i_tnr == 0.0;
            vi_i_tg4 == (vi_v_in-1.5)*4.0e-5;
      END USE;

ELSE
      vi_i_saida == 0.0;
      vi_i_tnr == 0.0;
      vi_i_tg4 == 0.0;
END USE;

break on vi_d_cin, vi_d_cnr, vi_d_cg4;
```

Pode-se perceber que todas as combinações possíveis de sinais de controle são abordadas pela estrutura anterior, uma vez que `vi_d_cin` tem precedência sobre `vi_d_cnr`, que por sua vez tem precedência sobre `vi_d_cg4`. Ou seja, se `vi_d_cin = '0'`, as saídas independem das outras chaves; se `vi_d_cin = '1'` e `vi_d_cnr = '0'`, as saídas independem de `vi_d_cg4`.

4.1.2.2 – Modelo comportamental

O modelo comportamental do conversor V-I implementa sua funcionalidade, porém leva em consideração seu comportamento não-ideal, a partir da caracterização do bloco apresentada em [37] e [42]. Uma característica do V-I que difere do ideal é que, para a faixa de entrada especificada, a saída não chega a cobrir toda a faixa de ±100 µA. Para este modelo, considera-se o comportamento do bloco já com as alterações descritas na seção 4.1.1 – embora, como será visto no capítulo 5, esse não seja muito diferente do comportamento do circuito original. A corrente de saída fica na faixa entre -97,43 µA e 97,89 µA. Portanto, vê-se que não só a corrente não atinge a faixa desejada, como

não é exatamente simétrica em relação à metade da faixa de entrada.

Outra não-idealidade significativa é a influência da temperatura no comportamento do V-I, pois é um problema cuja solução, quando da elaboração do modelo, estava pendente. Mesmo dentro da faixa de 0 a 70 ºC, que é a faixa especificada para a operação do circuito, variações de temperatura deterioram o funcionamento do conversor, alterando sensivelmente sua faixa de saída.

Feitas essas considerações, a característica de entrada/saída no modelo comportamental, quando ligado, foi primeiramente descrita a partir de alterações feitas no modelo ideal, como mostrado abaixo. As quebras de linha, nas declarações simultâneas, não existem no código.

```
IF vi_v_in'above(1.5) USE
     vi_i_saida == (vi_v_in-1.5)
          *(1.0-(27.0-temp_c)*0.00018)
          *1.958e-4;
ELSE
     vi_i_saida == (vi_v_in-1.5)
          *(1.0-(27.0-temp_c)*0.00018)
          *1.948e-4;
END USE;
```

Os fatores `1.958e-4` e `1.948e-4` modelam com razoável fidelidade a resposta não-ideal do circuito (a diferença entre as faixas positiva e negativa de saída é representada pelo uso condicional do atributo `above`). O uso de `*(1.0-(27.0-temp_c)*0.00018)` é uma aproximação da influência da temperatura – o valor da porta genérica `temp_c` representa a temperatura em graus Celsius. Como o V-I foi projetado para temperatura ambiente de 27 ºC, se `temp_c = 27.0` na simulação, o comportamento não é alterado. O fator `1.8e-04` é uma aproximação para que os extremos de temperatura afetem a saída como verificado no circuito. Entretanto, essa aproximação não descreve suficientemente bem a estabilidade térmica do V-I: a mudança da temperatura desloca a corrente de saída em relação à metade da escala (como visto na Figura 4.1, onde são mostradas as saídas do conversor para toda a faixa de entrada, sob temperaturas diferentes), o que não acontece na modelagem acima.
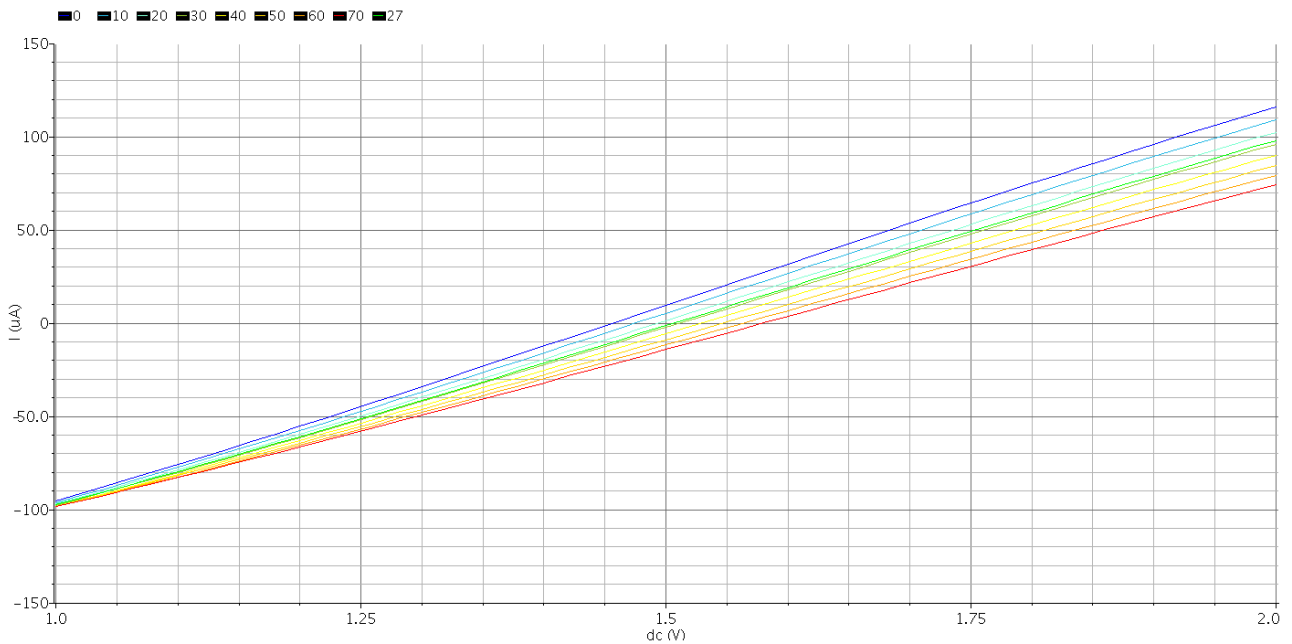
Figura 4.3 – resposta do V-I para diversas temperaturas

A solução encontrada foi desvincular-se do modelo ideal e descrever a resposta do circuito a partir das curvas mostradas na Figura 4.1. As curvas foram aproximadas por retas, a partir dos valores máximo e mínimo da saída em 0 ºC, 27 ºC e 70 ºC. Para minimizar a distorção que seria causada por uma aproximação com apenas uma equação de reta com coeficientes diferentes para cada temperatura (uma vez que as retas encontradas não convergem exatamente no mesmo ponto), foram definidas duas faixas: uma de 0 ºC a 27 ºC e outra de 27 ºC a 70 ºC. Descrevendo-se retas convergentes para estas faixas, chegou-se às seguintes equações, onde *i* é a saída de corrente, *v* é a entrada de tensão, *t* é a temperatura e o fator *coef* depende da temperatura:

$$i = [(v - 0{,}86801)\cdot coef - 123{,}210]\cdot 10^{-06},$$
$$\text{onde} \quad coef = 195{,}32 + \frac{(27{,}0 - t)}{1{,}697} \quad \text{e} \quad 0\ ºC \leq t \leq 27\ ºC \tag{4.3}$$

$$i = [(v - 0{,}95189)\cdot coef - 106{,}827]\cdot 10^{-06},$$
$$\text{onde} \quad coef = 195{,}32 + \frac{(27{,}0 - t)}{1{,}915} \quad \text{e} \quad 27\ ºC \leq t \leq 70\ ºC \tag{4.4}$$

As equações acima foram modeladas no código VHDL-AMS pela declaração dos coeficientes

```
constant coef_t00_27 : real := (195.32 + (27.0 - temp_c)/1.697)*1e-06;
constant coef_t27_70 : real := (195.32 + (27.0 - temp_c)/1.915)*1e-06;
```

e a utilização das seguintes linhas de código:

```
IF temp_c > 27.0 USE
      vi_i_saida == (vi_v_in - 0.95189)*coef_t27_70 - 106.827e-06;
ELSE
      vi_i_saida == (vi_v_in - 0.86801)*coef_t00_27 - 123.210e-06;
END USE;
```

A utilização das chaves para teste é análoga ao descrito para o modelo funcional, e por isso foi omitida aqui. É mostrada no código completo, apresentado no Apêndice E. Os resultados obtidos pela modelagem descrita são apresentados no capítulo 5.

4.1.2.3 – Modelo estrutural

A estrutura interna descrita no modelo estrutural é aquela mostrada na Figura 2.8, reapresentada na Figura 4.4 para facilitar referências. Os quatro sub-blocos foram descritos como entidades separadas, instanciadas por uma entidade hierarquicamente superior, que as instancia e conecta. Serão descritos primeiro os modelos dos sub-blocos.
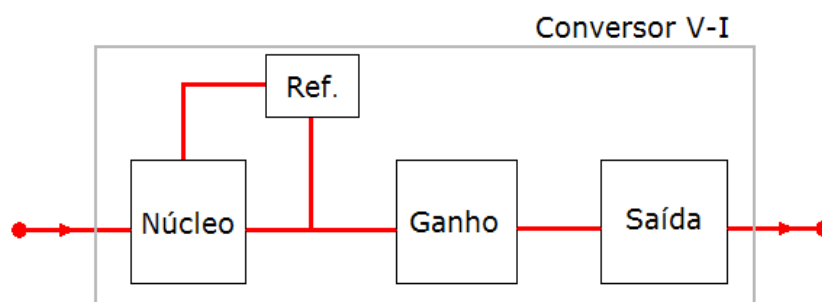
Figura 4.4 – diagrama do conversor V-I

Núcleo de conversão

O núcleo de conversão é implementado da mesma maneira que o modelo ideal do V-I, uma vez que funciona analogamente, porém com a necessidade de uma corrente constante (fornecida

pela referência) e uma faixa de saída de 2,5 a 12,5 µA. Assim, o conteúdo principal do módulo é o trecho de código abaixo, onde

- `vi_nuc_i_in` é a corrente de 1,84 µA necessária ao funcionamento do núcleo
- `vi_nuc_vin` é a tensão de entrada
- `vi_nuc_i_out` é a corrente de saída
- `nuc_out` é a corrente em um terminal interno

```
IF vi_nuc_i_in > 1.5e-06 and vi_nuc_i_in <2.0e-06 USE
      nuc_out == ((vi_nuc_vin)-1.0)*10.0e-06 + 2.5e-06;
      vi_nuc_i_out == nuc_out;
ELSE
      nuc_out == 0.0;
      vi_nuc_i_out == nuc_out;
END USE;

break on vi_nuc_i_in'above(1.5), vi_nuc_i_in'above(2.0);
```

O terminal `nuc` é definido dentro da arquitetura (ou seja, não faz parte da interface), e é utilizado apenas para definir uma corrente interna `nuc_out` que será utilizada para que o terminal de saída do núcleo não fique conectado ao terminal de entrada. Assim, é possível definir resistência de entrada e resistência de saída diferentes.

A condição de funcionamento mostrada acima considera uma faixa para a corrente `vi_nuc_i_in` entre 1,5 e 2,0 µA, dentro da qual funciona normalmente. Na realidade, o funcionamento do conversor não é interrompido de forma abrupta assim, mas sim deteriorado quanto mais longe `vi_nuc_i_in` estiver do valor ideal de 1,84 µA. Considerou-se uma faixa dentro da qual o erro é menos significativo, e fez-se essa aproximação.

Referência de corrente

A implementação da referência de corrente utiliza duas portas genéricas, uma porta de interface, e um terminal, conforme a declaração da entidade:

```
entity vi_rc is
```

```
generic (  vi_rc_i1 : real := 1.84e-06;
           vi_rc_i2 : real := -7.5e-06);
port (     quantity vi_rc_iout1 : out current;
           terminal vi_rc_t2 : electrical);
end entity vi_rc;
```

Onde `vi_rc_i1` é o valor da corrente que deve ser fornecida ao núcleo, e `vi_rc_i2` é o valor que deve ser drenado do nó interligando núcleo, refererência e estágio de ganho. A corrente através do terminal `vi_rc_t2`, que será conectado ao nó intermediário, é definida na arquitetura:

```
quantity vi_rc_out2 through vi_rc_t2;
```

Basta então associar os valores das portas genéricas às correntes de saída:

```
vi_rc_iout1 == vi_rc_i1;
vi_rc_out2 == vi_rc_i2;
```

Estágios de ganho

A implementação dos estágios de ganho (por simplicidade, o intermediário será chamado de *G4*, e o de saída, *G5*) é basicamente a mesma, apenas modificando a relação entrada/saída. Suas arquiteturas são definidas simplesmente por:

```
vi_g4_i_out == ganho*vi_g4_i_in;
```

para G4, e

```
vi_g5_i_out == ganho*vi_g5_in;
```

para G5. Nas equações acima, a constante real `ganho` é igual a 4,0 na arquitetura de G4, e a 5,0 na arquitetura de G5. Seria possível escrever as equações numericamente, porém preferiu-se o uso de constantes para facilitar alterações e testes.

Superbloco

O "superbloco" é uma entidade hierarquicamente superior (de nome *vi_macrobloco*) que instancia os sub-blocos e tem interface externa equivalente ao conversor V-I. Internamente, a

conexão entre os módulos é feita por um *port map*, cuja estrutura (onde o trecho entre colchetes é opcional) é:

```
[ <identificador (opcional)> : ] entity <biblioteca>.<entidade> (<arquitetura>)
        generic map  (<porta genérica do sub-bloco> => <porta genérica do superbloco>)
        port map  ( <porta do sub-bloco> => <porta do superbloco>);
```

Note-se que as portas genéricas são sempre valores constantes. Se uma porta genérica tiver um valor inicial definido na declaração da entidade e outro valor atribuído pelo mapeamento, o último substitui o inicial. Por outro lado, se uma porta genérica tiver valor inicial, o *generic map* não precisa ser declarado.

O trecho abaixo exemplifica o uso de um *port map*. Instancia o núcleo e conecta suas portas a terminais e quantidades declarados no código do bloco superior:

```
vi_nuc : entity work.vi_nuc(vi_nuc)
      port map (  vi_nuc_input => vi_v_in,
                  vi_nuc_i_in => vi_int_nr,
                  vi_nuc_output => vi_int_g4_in);
```

Isso é feito para todos módulos. O superbloco não precisa implementar nenhuma funcionalidade além da conexão entre os módulos, embora haja declarações simultâneas para satisfazer ao número exigido de equações simultâneas, como explicado no Apêndice A, e para realizar a conexão entre núcleo, referência e G4. Esta foi feita utilizando terminais internos, da seguinte maneira:

1. foram declarados terminais internos `vi_int_n_out` e `vi_int_g4_in`;
2. definiu-se a corrente `vi_int_ng4` de um terminal a outro, ou seja:

```
quantity vi_int_ng4 through vi_int_n_out to vi_int_g4_in;
```

3. declarou-se o terminal interno `vi_int_r_out2`;
4. definiu-se a corrente `vi_int_rg4`, saindo da referência e entrando em G4:

```
quantity vi_int_rg4 through vi_int_r_out2 to vi_int_g4_in;
```

Então, a declaração simultânea abaixo resulta em que a entrada de G4 seja a soma das correntes vindas do núcleo e da referência:

```
vi_int_g4_i_in == vi_int_ng4 + vi_int_rg4;
```

4.1.3 – Documentação

A estrutura da documentação sobre o V-I como IP é apresentada no Apêndice C. O conteúdo é apontado e exemplificado, porém não é reproduzido inteiramente no Apêndice C pois grande parte já foi apresentada em outras partes deste trabalho. Partiu de informações previamente existentes sobre o projeto, nas referências, e de resultados de simulação do bloco adaptado e dos modelos de alto nível. Não há, ainda, informações suficientes em relação à caracterização dos protótipos.

4.2 – CONVERSOR A/D

O projeto do conversor A/D já inclui estruturas de teste e observabilidade, e seu funcionamento pode ser controlado com bastante versatilidade através dos sinais digitais de controle. Portanto, não foram necessárias mudanças nos circuitos do bloco; a aplicação da metodologia consistiu de uma padronização da nomenclatura, identificação, modelagem e documentação. Os resultados são descritos no capítulo 5; a modelagem e documentação são descritas em detalhes nas seções a seguir.

4.2.1 – Modelagem em alto nível

Os modelos do A/D implementam de fato um conversor cíclico (não apenas realizam conversão de valores) com as características do bloco. Foram construídos modelos de níveis diferentes, de acordo com a metodologia proposta. Neste capítulo, são explicados os principais trechos de código utilizados para implementar a funcionalidade de cada modelo; o código completo encontra-se no Apêndice E. Resultados de simulação são apresentados e discutidos no capítulo 5.

4.2.1.1 – Modelo funcional ideal

No modelo funcional ideal, o A/D tem apenas um terminal elétrico (a entrada do bloco) e entradas e saídas binárias. Na declaração da arquitetura, é definido um subtipo `out_byte`, definido como um vetor de 8 bits, pela linha abaixo:

```
subtype out_byte is bit_vector(7 downto 0);
```

A arquitetura inclui um processo para modelar o funcionamento do A/D, onde são definidas algumas variáveis:

```
variable output_byte : out_byte;
variable i_smp : real;
variable i_smp_reg : real;
variable loop_count : integer := 1;
```

O vetor de 8 bits `output_byte` é uma variável que irá armazenar o resultado da conversão da entrada em uma palavra digital. `i_smp` é a variável que recebe o valor da entrada de corrente, ao início da conversão; como o valor de `i_smp` é alterado durante a conversão, a variável `i_smp_reg` apenas registra o valor amostrado, para referência durante a simulação. Por fim, `loop_count` representa uma contagem da quantidade de conversões que o A/D realizou desde que foi ligado, e seu uso será esclarecido à frente.

Quando ligado, o A/D realiza ciclos de conversão um após o outro. Por isso, usa-se um *loop* **WHILE**, repetindo o ciclo de conversão enquanto o conversor estiver ligado:

```
amostragem_e_conversao : WHILE ad_d_on_off = '1' LOOP
     -- ...
     -- <linhas de código modelando o ciclo de conversão>
     -- ...
END LOOP amostragem e conversão;
```

O ciclo de conversão, conforme modelado, pode ser entendido por partes sucessivas. Primeiro, amostra-se a corrente de entrada, `ad_i_in`. Isso é feito simplesmente pelas instruções:

```
i_smp := -ad_i_in;
i_smp_reg := i_smp;
```

Assim, a variável i_smp assume o valor da entrada naquele instante; o sinal negativo é por uma questão de notação, uma vez que o A/D considera positivos sinais entrando no nó, porém o padrão em VHDL-AMS é que correntes saindo do nó sejam positivas.

Em seguida, checa-se a entrada, como indicado nas linhas de código abaixo. Se estiver acima ou abaixo da faixa especificada, a saída é forçada para '1' ou '0', respectivamente, e aquele ciclo é terminado – aguardando-se o tempo necessário para a amostragem seguinte. Do contrário, segue-se com o ciclo de conversão.

```
check_range : IF i_smp < (-ref_100) THEN
report "Entrada fora da faixa especificada" severity warning;
     ad_d_saida <= '0';
     wait for 20 us;

ELSIF i_smp > ref_100 THEN
report "Entrada fora da faixa especificada" severity warning;
     ad_d_saida <= '1';
     wait for 20 us;
ELSE
     ...
END IF check_range;
```

O uso dos comandos report servem para auxiliar a simulação, e obviamente não representam funcionalidade real do bloco.

A conversão do valor da entrada em uma palavra binária, conforme descrito na seção 2.3.1, é modelada pelo seguinte trecho de código:

```
conversao : FOR i IN 7 DOWNTO 0 LOOP
     IF i_smp >= ref_comp THEN
          output_byte(i) := '1';
```

```
            i_smp := i_smp*2.0 - ref_100;
        ELSE
            output_byte(i) := '0';
            i_smp := i_smp*2.0 + ref_100;
        END IF;
    END LOOP conversao;
```

No código acima, foram utilizadas constantes (`ref_comp` e `ref_100`) no lugar de valores numéricos. Isso foi feito para facilitar alterações arbitrárias no modelo, para observação do comportamento, e por facilitar a compreensão da relação entre os valores e a funcionalidade: `ref_comp` representa a referência do comparador, e `ref_100` a corrente que é somada ou subtraída do sinal dependendo da conversão anterior.

A saída serial do bloco, de 1 bit a cada ciclo de relógio, começando pelo MSB, é implementada no trecho abaixo:

```
    saida : FOR i IN 7 DOWNTO 0 LOOP
        wait until AD_D_CLK = '1';
        ad_d_saida <= output_byte(i);
    END LOOP saida;
```

Para que o ciclo completo dure 20 µs, o ciclo de saida acima só ocorre 8 ciclos de relógio antes do fim desse período. Como neste modelo simples é fácil prever a duração do ciclo a partir da funcionalidade, seu período pode ser garantido usando-se, antes do laço de saída, um comando **`wait`**, numérico ou definido em relação ao período do relógio, definido em uma constante de tempo `t_clk` (igual a `62.5 ns`). Os únicos comandos que tomam tempo na simulação são o laço de saída (que demora 8 ciclos de relógio) e a inicialização – pois o conversor amostra 1 ciclo de relógio após ser ligado. Além disso, pode ser notado que a duração do laço de saída depende da resolução do A/D – que pode ser parametrizada pela constante `ad_res`, no caso igual a 8. Portanto, se forem parametrizados o período do relógio e a resolução do A/D, os dois comandos abaixo se equivalem:

```
    WAIT FOR 19.4375 us;
    WAIT FOR real(320 - ad_res - 1)*t_clk;
```

O uso de `real` mostrado converte o valor inteiro dentro dos parênteses para um número real, para que seja possível multiplicá-lo por `t_clk`. Ao final do laço de conversão, a variável `loop_count` é incrementada em 1:

```
loop_count := loop_count + 1;
```

Isso é feito para que o primeiro ciclo de conversão seja identificado. Em funcionamento contínuo (como é o caso do modelo funcional sempre que estiver ligado), ao final de um ciclo o sinal digital `ad_d_eoc` é levado a '1' para indicar o término de uma conversão e o começo da próxima. Outro sinal útil na compreensão da simulação, omitido até agora, é o sinal `ad_d_report_out`, que assume valor '1' durante o ciclo de saída e '0' durante o resto do tempo.

4.2.1.2 – Modelo comportamental

A modelagem das características não-ideais do conversor A/D exige uma abordagem um pouco diferente da descrita até agora. Fatores não-ideais tipicamente associados ao comportamento de conversores analógico/digital (explicados a seguir) são usualmente representados por valores máximos ou, graficamente, por curvas características de entrada/saída [1] (ilustradas na Figura 4.5 para um conversor de 3 bits). Embora estas representações sejam úteis na especificação e avaliação de um bloco, é difícil inseri-las em um modelo ou mesmo obtê-las por simulação funcional (pois como o funcionamento do A/D é temporizado, i.e., executa operações em instantes determinados, não é possível gerar uma curva contínua de entrada *versus* saída). Associada a estas dificuldades, há o fato de que o conversor A/D estudado aqui se encontra ainda em fase de testes, portanto não foi ainda completamente caracterizado.

A solução encontrada foi inserir tais características no código VHDL-AMS de forma parametrizada. A parametrização foi planejada para que o valor 0 (na notação adequada ao tipo do parâmetro) em qualquer parâmetro significa que a característica associada é ideal. Assim, uma vez feita a caracterização do A/D, o modelo pode ser facilmente atualizado. O modelo foi simulado para diferentes configurações destes parâmetros, verificando-se seus efeitos no funcionamento do bloco. Para a visualização do impacto destes comportamentos não-ideais na curva de entrada/saída do A/D, o algoritmo correspondente ao modelo VHDL-AMS foi traduzido para códigos MatLab. Isto foi feito pois a simulação em VHDL-AMS permite visualizar a saída apenas em relação ao tempo,

enquanto a simulação com MatLab permite gerar gráficos de *saída X entrada* para toda a faixa de entrada.
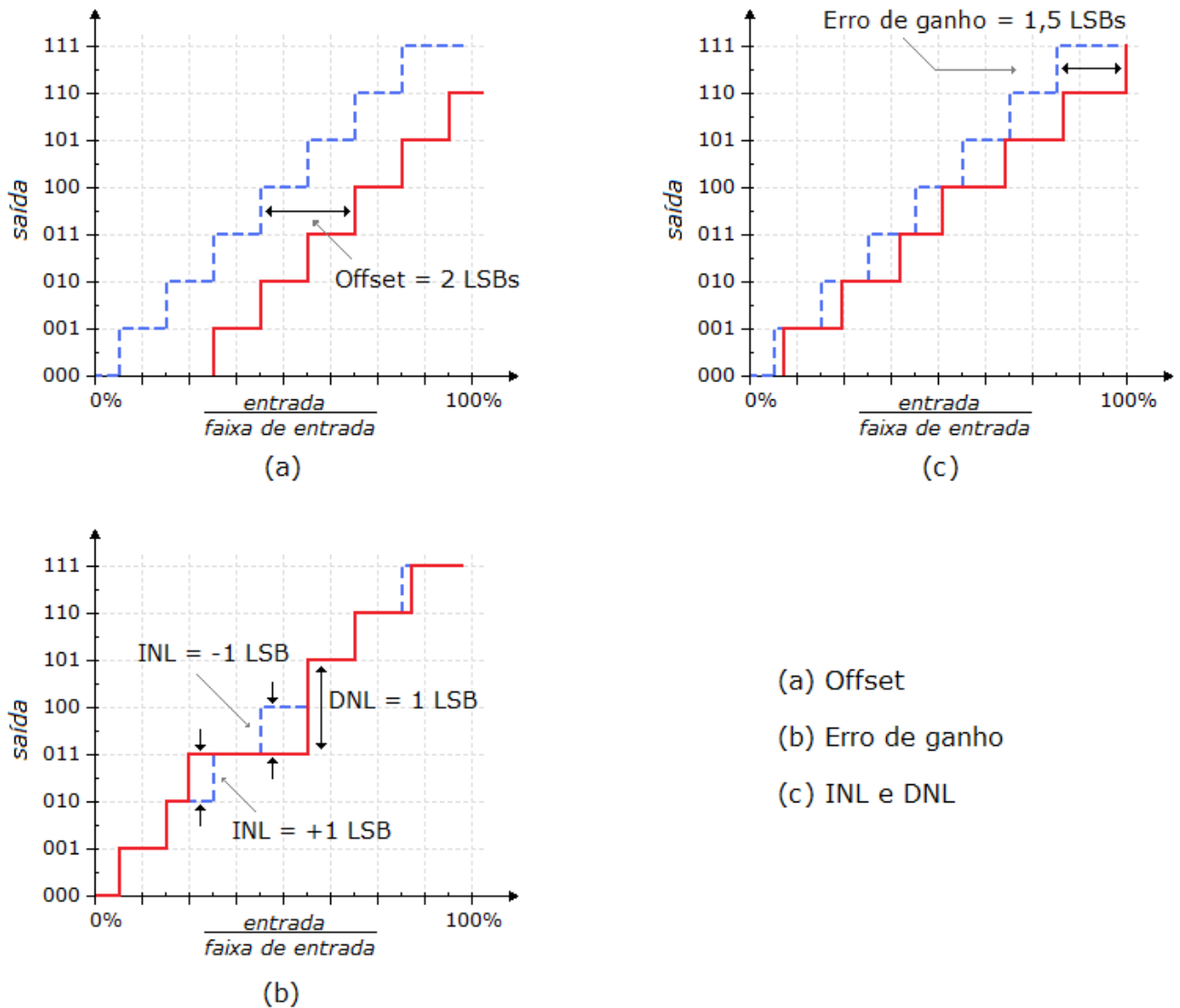
Figura 4.5 – efeitos de comportamento não-ideal em conversores A/D

Offset

É uma medida do deslocamento horizontal da curva de saída em relação à curva ideal, como pode ser visualizado na Figura 4.5(a) para um A/D de 3 bits. Costuma-se expressá-lo em LSBs, ou seja, em relação à menor variação da saída.

Na modelagem VHDL-AMS, foi modelado como um fator somado à corrente de entrada, no instante da amostragem. Nota-se que, desta maneira, o offset deve ter quantificado como corrente, não como número de bits. Isto foi abordado introduzindo no modelo a porta genérica

ad_offset_b – um número real representando o offset em LSBs – e criando-se uma variável interna ad_offset, de tipo real, que se relaciona à porta genérica ad_offset_b para representar o offset na escala das correntes do A/D. O fator de escala é igual à faixa de entrada dividido pelo número de níveis, ou seja, 200 μA/ 256 = $0,78125.10^{-6}$.

```
variable ad_offset : real := ad_offset_b*0.78125e-06;
```

Assim, sendo ad_i_in a corrente de entrada no A/D e i_smp a corrente amostrada, com a qual o A/D realizará o ciclo de conversão, a amostragem ideal é descrita como:

```
i_smp := -ad_i_in;
```

Ou seja, a corrente amostrada é exatamente igual à corrente de entrada naquele instante (o sinal negativo é para considerar a diferença na representação numérica entre uma corrente entrando no terminal e uma corrente saindo dele). A amostragem, considerando o offset, fica:

```
i_smp := -(ad_i_in + ad_offset);
```

Erro de ganho

O erro de ganho (*gain error*) é uma diferença entre as características de entrada/saída real e ideal, proporcional à magnitude de entrada. Pode ser visto como uma mudança na inclinação da reta ideal de resolução infinita. É medido pela diferença horizontal, em LSBs, entre a saída ideal e a saída real para o maior código de saída (no caso representado na Figura 4.5(b), 1 1 1).

No modelo comportamental do A/D, foi parametrizado por uma porta genérica ad_gerr_b, representando o erro de ganho um número de bits. Em seguida, criou-se uma variável interna ad_gerr:

```
variable ad_gerr : real := (128.0 - ad_gerr_b)/128.0;
```

Esta variável influencia a corrente utilizada na conversão, i_smp:

```
i_smp := -(ad_i_in)*(ad_gerr);
```

Desta maneira, no momento da amostragem aplica-se um ganho à corrente de entrada, igual a 1 se o erro de ganho for nulo (i.e., a corrente é amostrada corretamente), resultando em valor amostrado igual ao valor na entrada, e diferente de 1 se o erro de ganho for não-nulo, resultando em diferença entre os valores de `i_smp` e `ad_i_in`, diferença esta tanto maior quanto maior o valor absoluto da corrente de entrada. Deve-se notar que, como ilustrado na Figura 4.5(b), a definição usual do erro de ganho considera que ele se manifesta como um erro que aumenta com o aumento da entrada, sendo imperceptível para a menor entrada da escala; entretanto, como o A/D estudado aqui tem entrada variando em valores negativos e positivos, o erro de ganho se manifesta tanto no extremo inferior quanto no extremo superior da entrada, sendo imperceptível para entrada de corrente igual a 0 A. Por este motivo o valor para normalização utilizado na criação da variável `ad_gerr` foi 128, e não 256 (para que erro de ganho = 1 resulte em distorção de 1 bit em entrada máxima ou mínima).

Percebe-se que tanto o erro de ganho quanto o offset são modelados na linha que define a amostragem, porém separadamente, de forma que um valor nulo em qualquer dos dois parâmetros `ad_offset_b` e `ad_gerr_b` resulta em uma amostragem influenciada somente pelo outro parâmetro. A equação de `i_smp` no modelo comportamental, portanto, é:

```
i_smp := -(ad_i_in + ad_offset)*(ad_gerr);
```

INL

A não-linearidade integral (ou INL, de integral nonlinearity) pode ser vista como a diferença vertical máxima entre a curva ideal e a curva real. Simplificadamente, é definida como a diferença entre a palavra digital obtida e a palavra esperada, para dada entrada. A INL costuma ser quantificada em porcentagem ou em LSB – neste caso, só pode ser um número inteiro, já que é a diferença entre valores digitais. Uma dificuldade na modelagem da INL é que, por ser quantificada como a distorção máxima na curva, simplesmente o valor da INL, como pode ser encontrado no conjunto de especificações de um A/D, não permite que a curva do modelo seja alterada de maneira a corresponder fielmente ao comportamento real. Por exemplo, uma distorção de 2 bits na saída ocorrendo quando a entrada esta próxima ao extremo superior de sua faixa significa *INL = 2*; porém se a distorção ocorrer em qualquer outro ponto da curva, ainda assim tem-se *INL = 2*. Além disso, podem ocorrer outras distorções de 1 ou 2 bits ao longo da curva de entrada/saída, e elas não influenciam nesta quantificação da distorção máxima. Por isso, no modelo VHDL-AMS do A/D, foram criadas três portas genéricas: `ad_INL`, `ad_INL_vlow` e `ad_INL_vhigh`. A associação dos

três valores permite modelar uma distorção na característica de entrada/saída, da seguinte maneira:

- `ad_INL` é o valor da INL, em LSBs;
- `ad_INL_vlow` define um limite inferior, na entrada, para a ocorrência da distorção;
- `ad_INL_vhigh` define um limite superior, na entrada, para a ocorrência da distorção.

Caso ocorram distorções em mais de uma faixa da curva, a INL será apenas a maior distorção, porém as demais podem ser modeladas criando-se outros conjuntos equivalentes de três portas genéricas, e reproduzindo-se, no código, as linhas que modelam a ocorrência da distorção. Para uma única ocorrência, são as seguintes (as quebras de linha visam facilitar a leitura e não são utilizadas no código):

```
IF ad_INL /= 0   and (i_smp_reg > ad_INL_vlow)
                 and (i_smp_reg < ad_INL_vhigh) THEN

              i_smp := ad_INL_vlow + real(ad_INL)*0.78125e-06;
END IF;
```

A corrente `i_smp_reg` é uma variável gerada no mesmo instante de `i_smp`, com o mesmo valor, mas que não é modificada durante a conversão. Foi incluída no código pois `ad_i_in` varia continuamente, e `i_smp` é manipulada durante a geração da palavra digital de saída, assim `i_smp_reg` é um registro, durante todo o ciclo, da entrada no instante da amostragem.

As linhas acima aparecem no código após a amostragem, mas antes dos ciclos de conversão. Ou seja: se a entrada estiver na faixa definida pelo usuário (a partir de informações de caracterização), a corrente i_smp é considerada como tendo o valor que resulta na palavra digital incorreta. Como a medida da INL já implica saber qual a saída para determinada faixa de entrada, o erro de ganho e o *offset* podem de fato ser ignorados neste ciclo. Note-se também que a faixa determinada só é considerada caso o parâmetro `ad_INL` seja diferente de zero.

DNL

A não-linearidade diferencial (ou *differential nonlinearity*) pode ser considerada como a maior separação, em passos verticais, entre códigos adjacentes, representada pela equação (4.1) [1].

$$DNL = (Dcx - 1) \text{ [LSB]} , \qquad (4.1)$$

Dcx representa o tamanho do passo vertical real, em LSB. Portanto, na curva ideal, onde a diferença entre níveis adjacentes de saída é sempre de 1 LSB, a DNL é sempre nula. Entretanto, se algum nível digital não acontece na operação do A/D, a DNL é positiva. (A DNL pode ser negativa caso o conversor seja não-monotônico, i.e., quando saída, em determinada faixa, diminui para um aumento da entrada.) Assim como acontece com a INL, uma dificuldade em modelar a DNL é que é quantificada por um valor máximo. Porém, também como acontece com a INL, a inclusão da DNL na curva característica do modelo supõe conhecer a curva na faixa em que a distorção ocorre; no caso da DNL, saber que determinada palavra na saída não ocorre. Na modelagem VHDL-AMS, portanto, a DNL foi modelada por duas portas genéricas, `ad_DNL_in` e `ad_DNL_out`, ambas vetores de 8 bits (correspondente à resolução do A/D), e pela inserção das seguintes linhas no código:

```
IF output_byte = ad_DNL_in THEN
      output_byte := ad_DNL_out;
END IF;
```

Estas linhas ocorrem após o resultado da conversão ter sido armazenado na variável interna `output_byte`, mas antes que seja lido para ser apresentado na saída digital. Assim, o que o trecho de código acima faz é simplesmente verificar se o vetor que deve ser apresentado na saída é aquele 'subtraído' do comportamento real pela DNL. Se for, a saída apresenta outra seqüência de bits em seu lugar, de acordo com o o verificado na caracterização do bloco.

Pela descrição acima, nota-se outro problema na modelagem destes efeitos: embora sejam usualmente descritos separadamente, eles não ocorrem de maneira isolada. Se apenas o erro de ganho ocorre, se pode caracterizá-lo pelo deslocamento máximo da curva no fim da escala; porém, se ao mesmo tempo também há *offset*, o fim da escala não é uma leitura direta do erro de ganho. Isto significa que medir estes efeitos não-ideais isoladamente, modelá-los individualmente e então superpor os modelos pode causar distorções no comportamento final do modelo. Esta dificuldade, no entanto, já é abordada no método de modelagem proposto, pois cada efeito pode ter seu parâmetro definido separadamente. Com uma cautelosa caracterização e modelagem, que quantifiquem todos os quatro efeitos, levando em consideração o comportamento do circuito quando ocorrem simultaneamente, a configuração do modelo de alto nível seguirá a mesma

quantificação.

4.2.1.3 – Modelo estrutural

Para a elaboração do modelo estrutural, o A/D foi dividido em blocos funcionais como ilustrado na Figura 4.6:
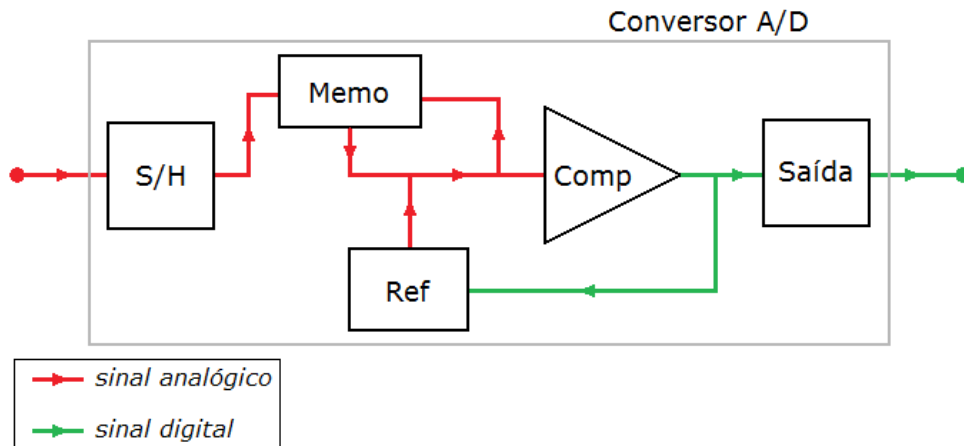
Figura 4.6 – diagrama do modelo estrutural do A/D

A estrutura acima representa a funcionalidade do A/D em ciclos. Os blocos agem da seguinte forma:

- O bloco *S/H* amostra a corrente de entrada e mantém o valor amostrado durante o período de conversão;
- O bloco *Memo* fornece a corrente que, somada àquela vinda de *Ref*, será utilizada por *Comp* na obtenção dos bits. No primeiro ciclo, a saída de *Memo* é igual à entrada (sinal amostrado em *S/H*); em cada ciclo posterior, o bloco usa a corrente da entrada de *Comp* no ciclo anterior, e a multiplica, sucessivamente, por 2, 4, 8, 16, 32, 64 e 128, um valor a cada ciclo;
- O bloco *Ref* soma corrente de +100 µA ou –100 µA, dependendo da saída digital de Comp no ciclo anterior (no primeiro ciclo, não soma nenhuma corrente);
- A soma dos sinais vindos de *Memo* e *Ref* é comparado com 0,0 no bloco *Comp*; caso seja maior ou igual, a saída digital de *Comp* é '1'; caso contrário, é '0';
- O bloco *Saída* recebe, de *Comp*, os 8 bits resultantes da conversão, e os armazena. Nos últimos 8 ciclos de relógio do período de 20 µA, os fornece na saída.

Há duas diferenças principais entre o modelo estrutural e a estrutura real do A/D: primeiramente, o A/D não tem um bloco funcionando como o bloco *Saída* na Figura 4.6; este apenas repete a saída do comparador, e foi incluído no modelo para que tempo do ciclo de conversão tome o tempo que toma no circuito. Outra diferença é que o bloco *Memo*, no modelo, é uma simplificação de uma estrutura de várias células copiadoras de corrente que, através de comandos sincronizados de chaves, lêem a corrente entrando no comparador a cada ciclo e dobram seu valor.

Essas diferenças, no entanto, não invalidam a modelagem, que, como dito anteriormente, é uma representação da estrutura interna que não precisa ser estritamente idêntica à do bloco, apenas representá-la para permitir um entendimento mais aprofundado. Todos os blocos têm uma entrada digital de sinal de relógio e outra que informa o fim do ciclo de conversão.

A seguir, as características principais dos sub-blocos será explicada, bem como do bloco hierarquicamente superior que os instancia e conecta (superbloco). Este não será detalhado, pois sua implementação é praticamente idêntica à do superbloco do modelo estrutural do conversor V-I, conectando os sub-blocos e mapeando a interface externa.

S/H

O bloco *S/H* simplesmente lê o valor de entrada (`ad_sh_iin`) a cada 20 μA, e dá uma saída correspondente. Como visto na descrição do modelo comportamental, é na amostragem que são modelados os efeitos de offset, erro de ganho e INL, que são modelados também neste bloco. O valor da variável de amostragem `i_smp` é obtido, em cada ciclo de conversão, pelas linhas:

```
i_smp := -(ad_sh_iin + ad_offset)*(ad_gerr);

INL : IF ad_INL /= 0
      and (i_smp_reg > ad_INL_vlow)
      and (i_smp_reg < ad_INL_vhigh) THEN

           i_smp := ad_INL_vlow + real(ad_INL)*0.78125e-06;

END IF INL;
```

A saída `ad_sh_saida` é modelada para que mantenha o valor de `i_smp` durante todo o

ciclo. Para que seja recalculada quando `i_smp` mudar, é lido o sinal de fim de conversão gerado pelo bloco *Saída*. As linhas que regem a saída de *S/H*, então, são:

```
ad_sh_saida == i_smp;
break on ad_sh_eoc;
```

É necessário observar que nos códigos anteriores do A/D, `i_smp` é uma variável, e portanto acessível apenas dentro do processo no qual é declarada (não sendo possível, portanto, fazer `ad_sh_saida == i_smp`). Aqui, é uma *variável compartilhada* (*shared variable*), declarada na arquitetura e podendo ser usada por mais de um (ou nenhum) processo.

Memo

O valor da saída de *S/H* é lido como entrada no bloco *Memo*, que a princípio o repete na saída `ad_memo_iout`; então, a cada ciclo de relógio, o bloco usa como entrada o valor da corrente entrando em *Comp* no período anterior, a saída é o dobro da entrada. Isso é implementado pelas linhas a seguir, onde `ad_i_ciclos` é uma variável compartilhada:

```
ad_memo_iout == ad_i_ciclos;
        ...

        i_memo_smp := - ad_memo_iin;

                ad_i_ciclos := i_memo_smp;
                wait until ad_memo_clk = '1';

        ciclos_memo : FOR i IN 1 TO (ad_res-1) LOOP
                ad_i_ciclos := i_memo_smp*2.0;
                wait until ad_memo_clk = '1';
        END LOOP ciclos_memo;
```

Assim, a saída é inicialmente igual à entrada, e depois, no *loop* `ciclos_memo`, o valor de `ad_i_ciclos` é atualizado a cada ciclo de relógio. Ao fim dos 8 ciclos da conversão, o bloco fica inativo aguardando o sinal de final de conversão ou reinicialização.

Ref

O bloco *Ref* soma ao valor de saída de *Memo* ±100 μA, dependendo do resultado da comparação realizada no ciclo anterior pelo bloco *Comp* (lido em `ad_ref_comp`). A saída (`ad_ref_iout`) é o resultado da multiplicação da variável compartilhada `ad_i_var`, que pode ser -1, 0 ou 1, por 100×10^{-06}:

```
ad_ref_iout == real(ad_i_var)*100.0e-06;
```

No primeiro dos 8 ciclos de conversão, a saída de *Ref* deve ser nula. O resultado da comparação é armazenado a cada ciclo; nos ciclos seguintes, é utilizado para definir o valor de `ad_i_var` conforme as seguintes linhas de código demonstram:

```
FOR i IN 1 TO (ad_res) LOOP
    wait until ad_ref_clk = '1';
    CASE comparacao_anterior IS
        when '0' =>
            ad_i_var := -1;
        when '1' =>
            ad_i_var := 1;
    END CASE;

comparacao_anterior := ad_ref_comp;

END LOOP;
```

Ao fim dos 8 ciclos da conversão, o bloco fica inativo e aguarda o sinal de final de conversão ou reinicialização.

Comp

A função de *Comp* é simplesmente entregar na saída `ad_comp_saida` um bit '1' caso a entrada `i_comp_smp` seja igual ou maior do que 0, e '0' caso contrário. Além disso, o bloco também tem uma saída digital, `ad_comp_report_out`, que, quando em nível alto ('1'), indica ao bloco *Saída* que os valores da conversão estão sendo entregues, um a cada ciclo de relógio. Isso é implementado pelas linhas:

```
ad_comp_report_out <= '1';
comparacao : FOR i IN 1 TO ad_res LOOP
      IF i_comp_smp >= 0.0 THEN
            ad_comp_saida <= '1';
      ELSE
            ad_comp_saida <= '0';
      END IF;
      wait until ad_comp_clk = '1';
END LOOP comparacao;
ad_comp_report_out <= '0';
```

Saída

O bloco de saída, ao receber o sinal de que a saída de *Comp* representa o resultado da conversão, os lê e armazena, na variável `output_byte`, um bit a cada ciclo de relógio. Nos últimos 8 períodos de relógio, entrega na saída (`ad_saida_out_bit`, que representa a saída do A/D) os bits lidos. Isso é implementado pelas seguintes linhas:

```
armazena : FOR i IN (ad_res -1) DOWNTO 0 LOOP
      output_byte(i) := ad_saida_in;
      wait until ad_saida_clk = '1';
      ad_saida_d_eoc <= '0';
END LOOP armazena;

wait for real((320 - (2*ad_res - 1)))*t_clk;
ad_saida_report_out <= '1';

saida: FOR i IN (ad_res -1) DOWNTO 0 LOOP
      wait until ad_saida_clk = '1';
      ad_saida_out_bit <= output_byte(i);
END LOOP saida;
ad_saida_report_out <= '0';
```

Além disso, o bloco gera um sinal digital indicando o fim da conversão, e implementa, se necessário, a DNL; ambas as funcionalidades equivalem às do A/D como bloco único, como foi descrito nos modelos funcional e comportamental do A/D, por isso não serão reapresentadas aqui.

Superbloco

O bloco que cerca as estruturas descritas anteriormente pode simplesmente realizar as conexões entre eles, através de *port maps*, como descrito para o V-I no item 4.1.2.3. No caso do A/D, as interconexões funcionaram, porém ocorreram alguns problemas na implementação do bloco hierarquicamente a partir dos sub-blocos que afetaram a funcionalidade do modelo estrutural, como será discutido no capítulo 5.

4.2.2 - Documentação

Como no caso do V-I, a documentação do A/D foi elaborada seguindo a metodologia proposta, a partir de informações disponíveis previamente e coletadas durante a elaboração deste trabalho. É apresentada no Apêndice D.

5 – RESULTADOS E DISCUSSÃO

Neste capítulo serão apresentados os resultados dos procedimentos descritos no capítulo anterior. Na seção 5.1 são mostrados os resultados obtidos na adaptação e modelagem do conversor V-I, e na seção 5.2, do conversor A/D.

5.1 – CONVERSOR V-I

Na seção 5.1.1, são apresentados os resultados obtidos na adaptação do circuito original do V-I à metodologia, conforme descrito no capítulo 4. Na seção 5.1.2, são mostrados os resultados obtidos com os modelos VHDL-AMS.

5.1.1. - Adaptação do circuito à padronização de IP

A inserção de estruturas de observabilidade e controlabilidade gerou os resultados apresentados nesta seção, todos de simulações elétricas, que visam principalmente avaliar seu impacto no desempenho do circuito.

5.1.1.1 – Distorção nos sinais causada pelas chaves

A adição das chaves e pinos como estruturas de testabilidade foi implementada com transistores PMOS e NMOS conforme descrito na seção 4.1.1. Na Figura 5.1 são mostrados resultados de simulação em que o circuito original e o circuito após a adaptação foram simulados juntos.
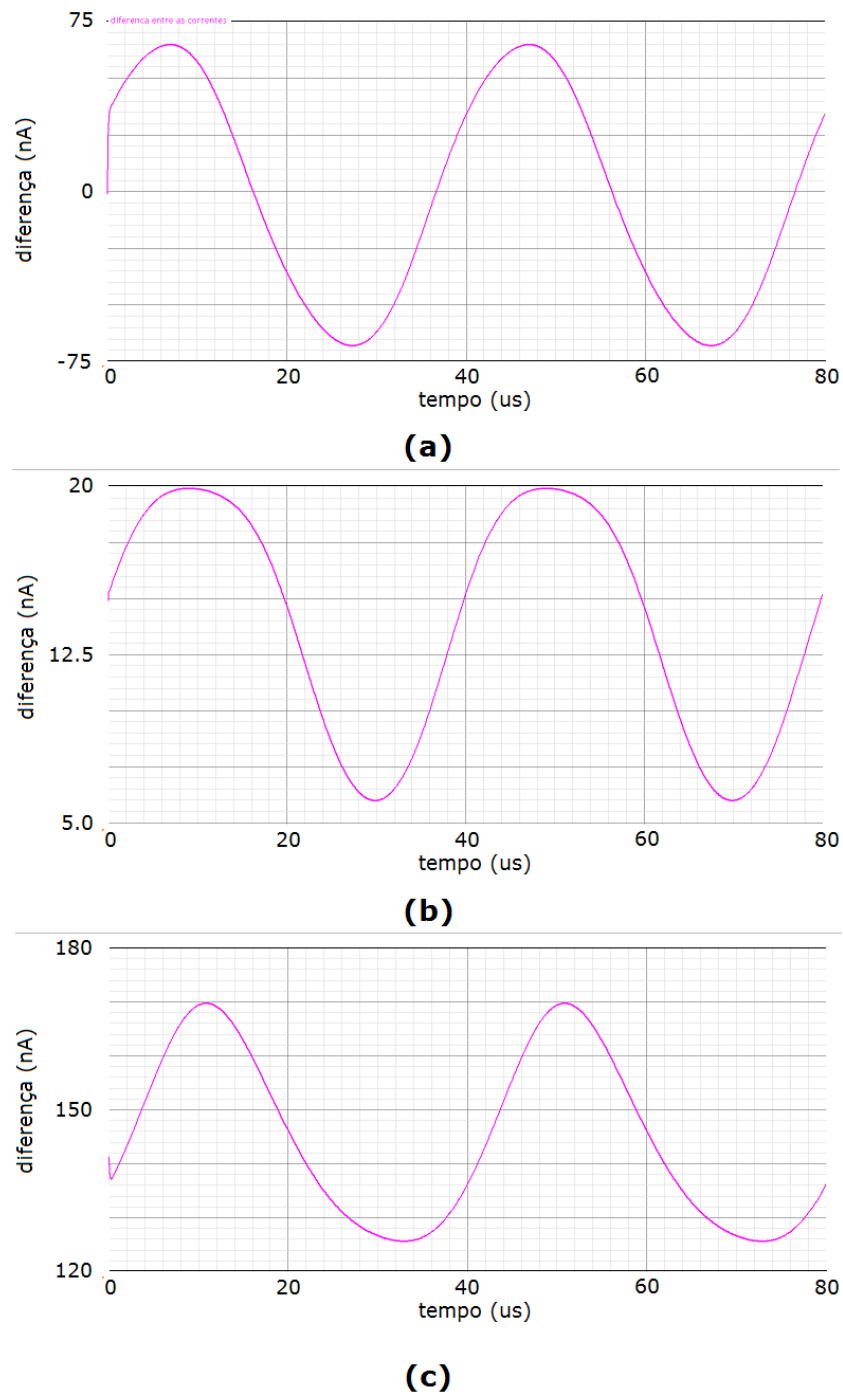
Figura 5.1 – diferença entre correntes esperadas e obtidas nos pinos (a) de saída; (b) t_nr; (c) t_g4

Foram observadas as correntes de saída, teste NR (i.e., saída pelo pino *t_nr*) e teste G4 (saída pelo pino *t_g4*). As ondas mostradas na Figura são os cálculos da diferença entre a corrente no circuito original e a corrente no pino de teste. A entrada de tensão é senoidal de 25 kHz, varrendo a faixa de 1 a 2 V a cada 40 µs.

Vê-se que o modo de teste cumpre seu objetivo, pois embora as chaves influenciem as correntes no circuito, as diferenças entre correntes esperadas e obtidas são muito pequenas. A maior diferença medida, em valor absoluto, é de 170 nA (entre a corrente na saída do estágio intermediário de ganho e a corrente medida pelo pino *t_g4*), o que corresponde a menos de 0,43% da faixa de correntes para esse ponto do circuito.

Embora a corrente na chave CIN não seja relevante, a queda de tensão pode ser, uma vez que distorce a entrada, levando o núcleo a converter um valor diferente do efetivamente entregue pelos sinais externos ao conversor. Essa queda de tensão foi medida, durante a operação com a chave conduzindo, verificando-se uma variação de -3,12 µV a 3,27 µV (mostrada na Figura 5.2) – desprezível em comparação com a faixa de entrada de 1 V.
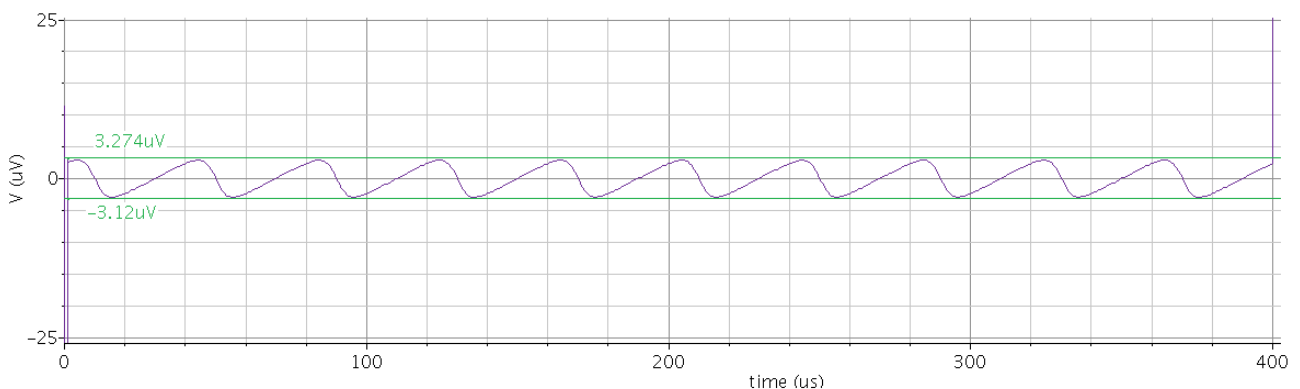
Figura 5.2 – queda de tensão na chave CIN durante operação

5.1.1.2 – Estabilidade térmica

Outro ponto a se observar na adaptação do V-I, quanto à degradação do desempenho, é em relação à influência da temperatura, que é um dos fatores mais problemáticos do projeto original do bloco. A introdução das chaves, entretanto, não afetou negativamente o circuito em relação à temperatura. O gráfico mostrado na Figura 5.3 representa as respostas do V-I em toda a faixa de

entrada, para temperaturas entre 0 ºC e 70 ºC.

Embora a tolerância à temperatura não seja ótima, no circuito adaptado, não representa piora significativa em relação aos resultados anteriores. A Tabela 5.1 compara os resultados para temperaturas de 0 ºC , 27 ºC e 70 ºC nos extremos da faixa de entrada.
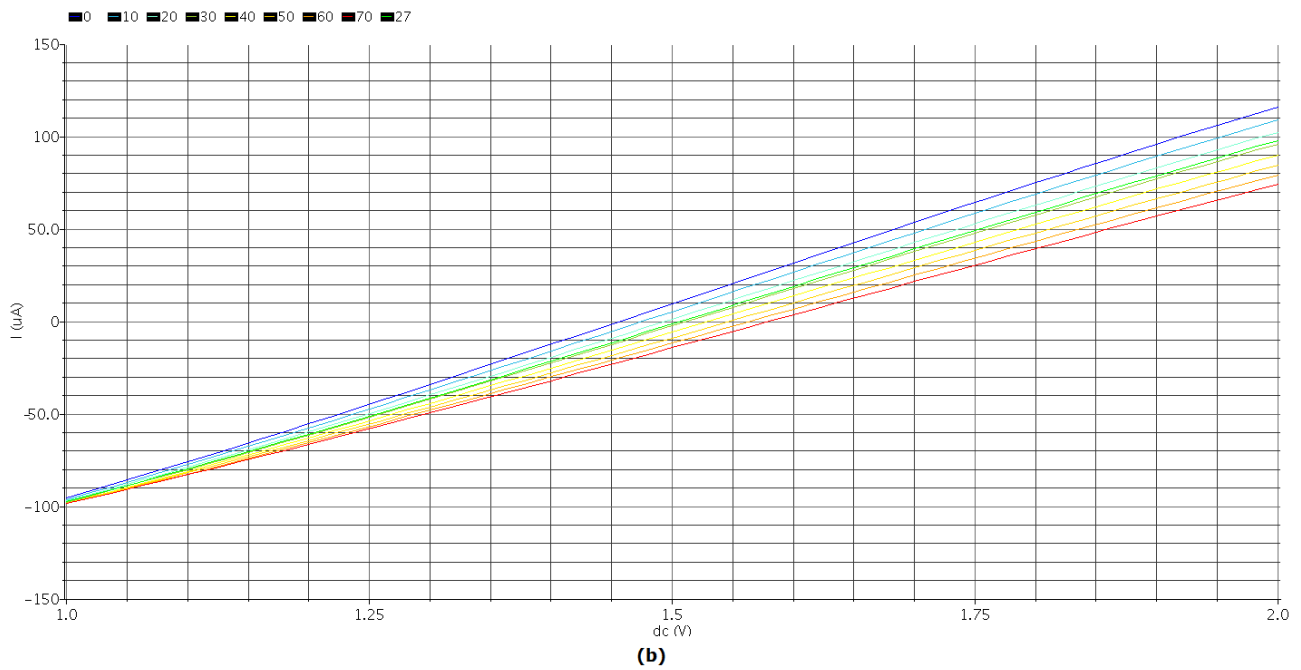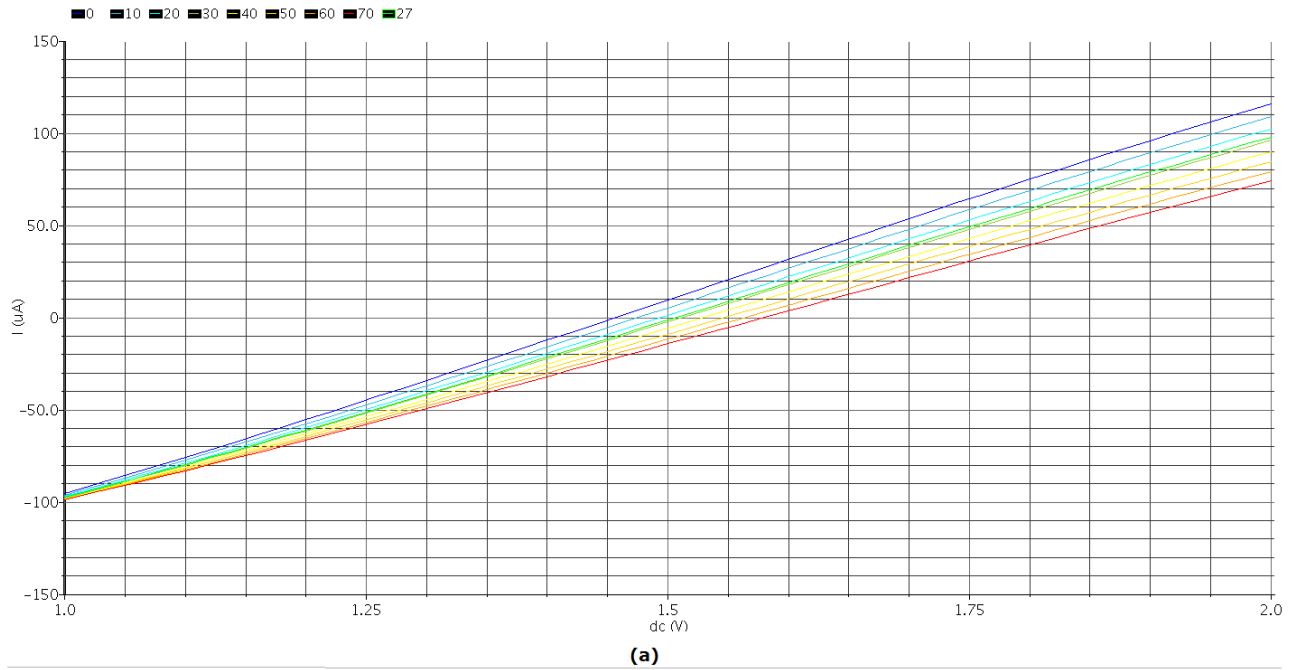
Figura 5.3 – tolerância à temperatura do conversor (a) original e (b) adaptado

Tabela 5.1 – tolerância à temperatura do conversor original e adaptado

	Corrente mínima (µA)			Corrente máxima (µA)		
	0 ºC	**27 ºC**	**70 ºC**	**0 ºC**	**27 ºC**	**70 ºC**
V-I original	-95,39	-97,49	-98,58	116,0	97,95	74,41
V-I adaptado	-95,33	-97,43	-98,51	115,9	97,89	74,36

5.1.1.3 – Combinações de sinais de controle

Os resultados de simulações do V-I para as diferentes combinações dos sinais de controle é mostrado na Figura 5.4. Embora, pelo tamanho da Figura, as escalas não sejam bem visíveis, podem ser vistos os estados previstos para a operação do V-I. Em particular, as três primeiras combinações – 1 1 1, 1 0 1 e 1 1 0 – mostram, respectivamente, a operação normal do conversor, o teste NR e o teste G4.
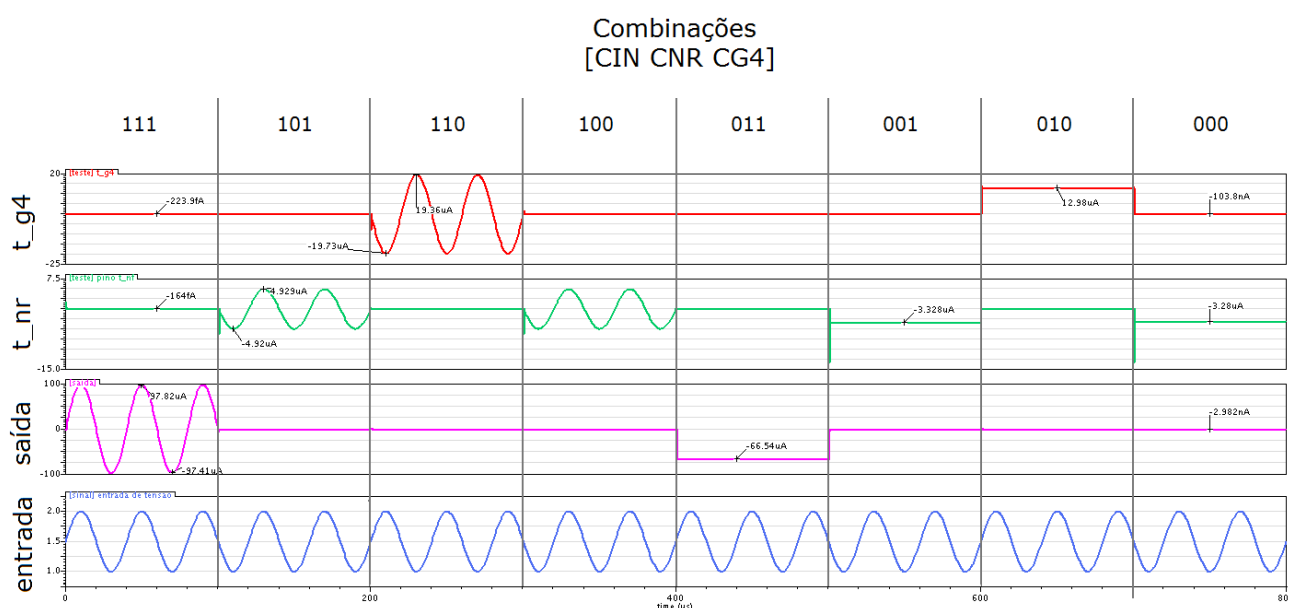
Figura 5.4 – simulação do V-I para todas combinações de chaves

Por esta simulação, pode-se construir a Tabela 5.2, com detalhes sobre as combinações de chaveamento no circuito. Todos os valores de corrente aproximados por “0” na Tabela são menores do que 0,2 µA. Os estados de operação previstos estão destacados.

Os resultados mostram que a implementação de um “desligamento” não é completamente satisfatória; a saída é zerada e o consumo é diminuído, porém o conversor continua consumindo

potência e há uma pequena saída de corrente no pino *vi_t_nr*. De fato, foram feitas tentativas de implementar um desligamento efetivo do circuito – utilizando chaves entre o circuito e sua alimentação elétrica. Essa alternativa, apesar da vantagem de diminuir muito mais drasticamente as correntes durante o desligamento, é muito drástica para circuitos analógicos como o V-I, pois mesmo durante o estado "ligado" altera as tensões no circuito de maneira significativa. Se fosse implementada, isso exigiria refazer o dimensionamento de todos os sub-blocos, que já é sensível, muito provavelmente impondo fortes restrições de desempenho.

Tabela 5.2 – modos de operação do conversor V-I alterado

CIN	CNR	CG4	Operação	Saída	Pino t_nr	Pino t_g4
0	**0**	**0**	Desligamento	0	-3,3 µA	0
0	**0**	**1**		0	-3,3 µA	0
0	**1**	**0**		0	0	13 µA
0	**1**	**1**		-66,5 µA	0	0
1	**0**	**0**		0	Faixa de ±5,0 µA	0
1	**0**	**1**	Teste (NR)	0	faixa de ±5,0 µA	0
1	**1**	**0**	Teste (G4)	0	0	Faixa de ±20,0 µA
1	**1**	**1**	Normal	Faixa de ±100,0 µA	0	0

Nota-se ainda pelos resultados acima que, conforme já citado na descrição dos modelos, a chave *CNR* tem precedência sobre *CG4*, i.e., se *CNR = 0* o circuito está em modo de teste NR, mesmo se *CG4* também estiver em 0. Porém, o estado (1 0 1) foi preferido por alterar apenas uma chave em relação à operação normal.

Pela Figura 5.4, também percebe-se a ocorrência de picos de corrente em alguns instantes de transição. Na transição de (1 1 1) para (1 0 1), por exemplo, há uma breve corrente negativa em *t_nr*. Estes picos acontecem pois há pequenas faixas de tensão na chave que fazem ambos os transistores P e N conduzirem. Tais picos, porém, não foram considerados importantes, pois a princípio as chaves simplesmente selecionam um modo de operação, então as transições não são críticas. Até por isso, os sinais de controle, na simulação, não foram estritamente rigorosos quanto a sobreposição ou tempos de subida e descida.

5.1.1.4 – Quedas de tensão nas chaves

As quedas de tensão nas chaves durante a operação do circuito foram medidas, bem como as correntes. Na Figura 5.5 e na Tabela 5.3, a notação é a seguinte:

- *Ron* é a resistência da chave conduzindo;
- *Roff* é a resistência da chave em corte;
    - *CNR* e *CG4* identificam os conjuntos das chaves de teste, de acordo com o uso feito até agora; os índices *n* e *p* indicam o transistor NMOS ou PMOS da chave em questão.

Tabela 5.3 – resistências equivalentes das chaves de teste

	CNRn	**CNRp**	**CG4n**	**CG4p**
R_{ON} (min)	700 Ω	3,74 kΩ	600 Ω	3,4 kΩ
R_{ON} (max)	850 Ω	4,02 kΩ	890 Ω	4,6 kΩ
R_{OFF} (min)	$11{,}6.10^{15}$ Ω	$1{,}2.10^{16}$	$7{,}0.10^{15}$ Ω	$6{,}0.10^{15}$ Ω
R_{OFF} (max)	$17{,}2.10^{15}$ Ω	$3{,}7.10^{16}$	$3{,}0.10^{16}$ Ω	$6{,}2.10^{16}$ Ω

Figura 5.5 – resistência das chaves de teste

Na Figura 5.5 são mostradas as curvas obtidas na medição da resistência equivalente para as chaves, em toda a faixa de entrada do bloco, tanto conduzindo quanto em corte. Como mostrar todas as ondas em escala tomaria muito espaço, optou-se por apenas apresentar os gráficos lado a lado, e resumir as informações mais importantes na Tabela 5.3.

A resistência da chave *CIN* não é indicada pois a entrada do núcleo de conversão é a porta (*gate*) de um transistor MOS, o que já significa corrente extremamente baixa; o cálculo de tensão dividida pela corrente não gera ondas significativas. A resistência de *CIN* em corte foi calculada como sendo da ordem de 10^{16} Ω. Todas as outras chaves em corte têm resistência da ordem de 10^{15} Ω.

5.1.1.5 – Leiaute

Foi construído um leiaute da versão do circuito com a implementação das chaves, na tecnologia CMOS de 0,35 µm *C35B4C3*, da *Austria Microsystems*. Foi tomado cuidado para manter o leiaute o menor possível; mesmo com a inserção das chaves e novos pinos, foi possível manter o leiaute do V-I adaptado (mostrado na Figura 5.6, com indicações dos blocos) do mesmo tamanho do original, de 100,28 µm x 217,33 µm [37].

O leiaute foi construído já com a nomenclatura modificada dos pinos. Incluiu-se no leiaute também a etiqueta de identificação de IP proposta, em uma camada de texto (Figura 5.7). A etiqueta identifica os seguintes campos:

- Número de palavras-chave: 4
- Fornecedor: *LDCI-UnB*
- Produto: *conversor_vi*
- Versão: *1.1*
- Área: *0,218mm*^{2}
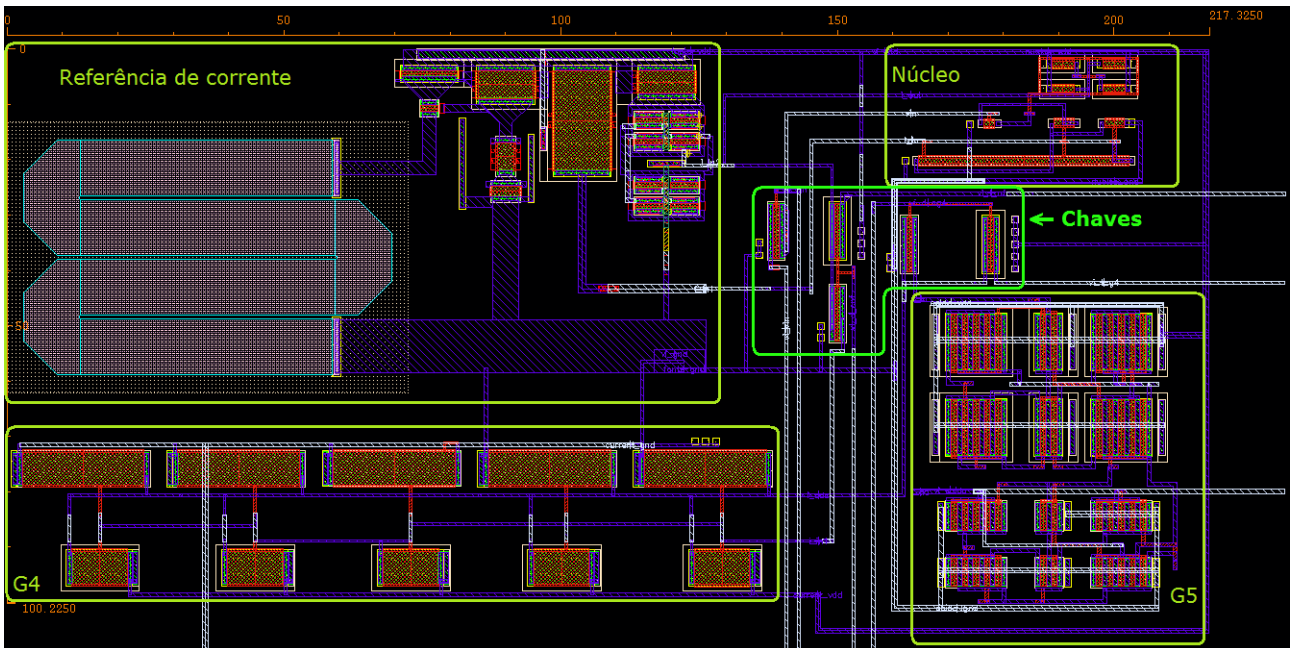
Figura 5.6 – leiaute do conversor V-I adaptado
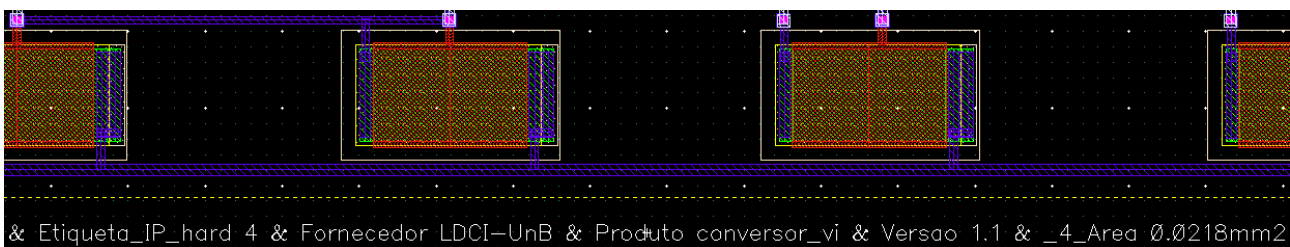
Figura 5.7 – etiqueta de IP no leiaute do V-I

5.1.2 - Modelagem

Os resultados de simulação da modelagem em VHDL-AMS serão apresentados separadamente, de acordo com os respectivos modelos.

5.1.2.1 – Modelo funcional

A simulação do modelo funcional gerou as saídas mostradas na Figura 5.8, validando o modelo para o código e *testbench* apresentados no Apêndice E. De cima para baixo, a primeira onda é o sinal de entrada de tensão (utilizou-se uma senóide de 22 kHz); em seguida, aparecem os três sinais digitais de controle; a onda destacada é a saída do bloco, e, abaixo, são a corrente de saída em *t_nr* e em *t_g4*.
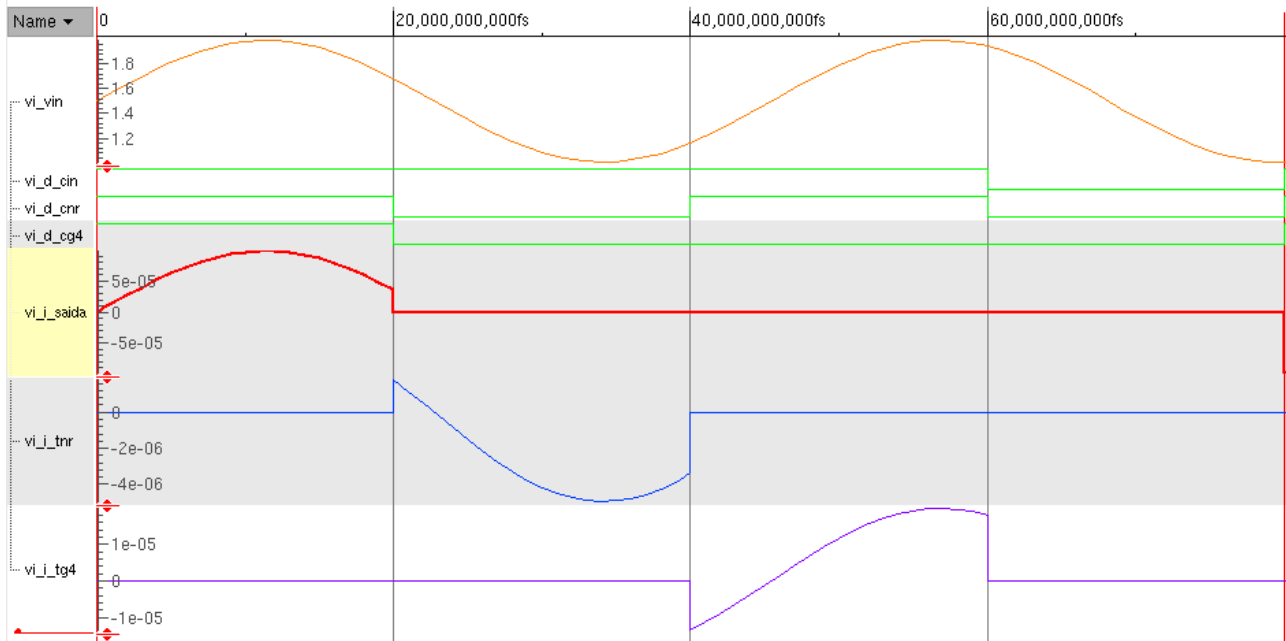
Figura 5.8 – simulação do modelo funcional do conversor V-I

Note-se que as três ondas de saída são apresentadas em escalas diferentes. Foram inseridos cursores verticais para indicar as transições dos sinais de controle – as quatro combinações mostradas na figura são, na ordem: (1 1 1), (1 0 0), (1 1 0) e (0 0 0); essas combinações representam todas as outras, conforme foi explicado na descrição do modelo.

5.1.2.2 – Modelo comportamental

As simulações do modelo comportamental do V-I corresponderam ao esperado. Para 27 ºC as diferenças entre a resposta do modelo funcional e a do modelo comportamental são sutis. Uma comparação da saída comportamental com a ideal é mostrada na Figura 5.9. As ondas na Figura 5.9(a) são a superposição dos resultados de simulação VHDL-AMS do modelo ideal (onda azul-escura, externa) e comportamental (onda verde, interna), demonstrando, para mesma entrada, os efeitos não-ideais no conversor. A Figura 5.9 (b), onde é mostrado o resultado de simulação elétrica do V-I sobreposto à saída ideal, é extraída de [37]. A comparação dos resultados ilustra a fidelidade da modelagem em relação ao circuito original.
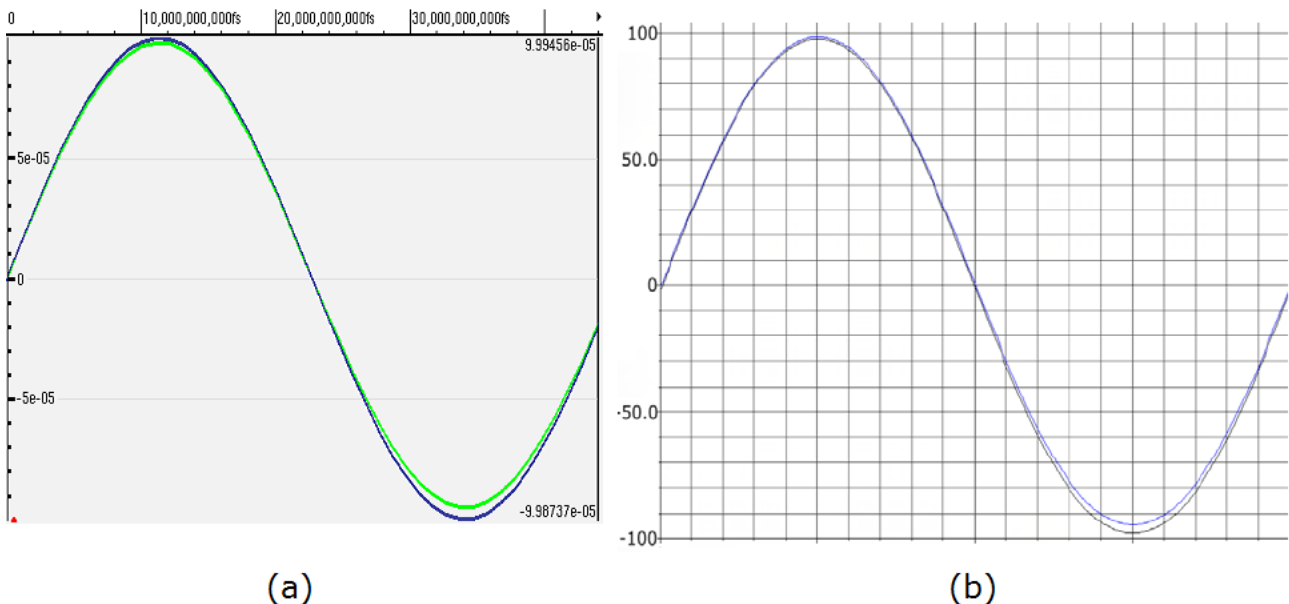
Figura 5.9 – resposta do V-I em 27 ºC comparada à ideal:
(a) no modelo comportamental; (b) no circuito

Quanto à estabilidade térmica, a simulação do modelo comportamental é apresentada na Figura 5.10. O resultado obtidos para a simulação das saídas para toda a faixa de tensão são equivalentes ao mostrado na Figura 5.3(b), para o circuito simulado, demonstrando a validade do modelo. Na Figura 5.10(b), é mostrado o funcionamento do modelo comportamental para diferentes combinações de chave e para diferentes temperaturas.

A Tabela 5.4 lista os principais valores obtidos pela modelagem, em comparação com os valores esperados (obtidos por simulação elétrica do circuito). Como descrito na seção 4.1, para temperaturas intermediárias entre os valores mostrados na Tabela 5.4 a aproximação usada na modelagem se desvia ligeiramente da real, porém pode ser visto pelos resultados que a modelagem é bastante adequada.

Tabela 5.4 – tolerância à temperatura do modelo comportamental do V-I

	Corrente mínima (μA)			**Corrente máxima (μA)**		
	0 ºC	**27 ºC**	**70 ºC**	**0 ºC**	**27 ºC**	**70 ºC**
Modelo comportamental	-95,33	-97,43	-98,51	115,9	97,89	74,36
Circuito	-95,33	-97,43	-98,51	115,9	97,89	74,36
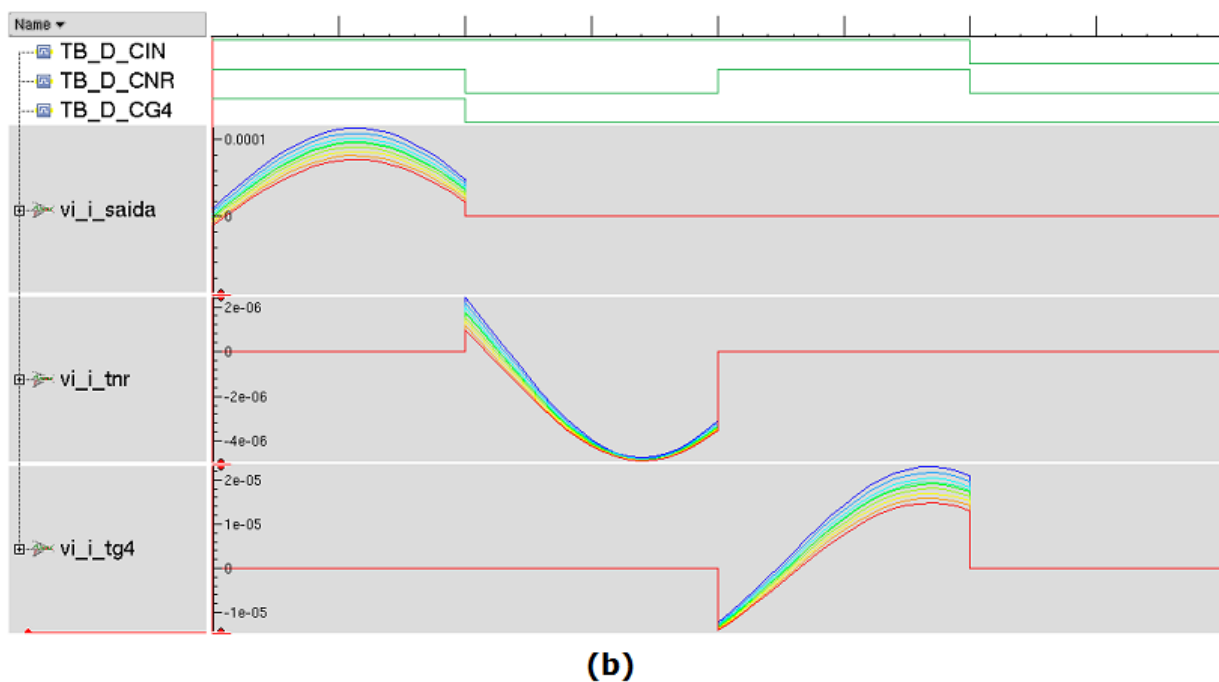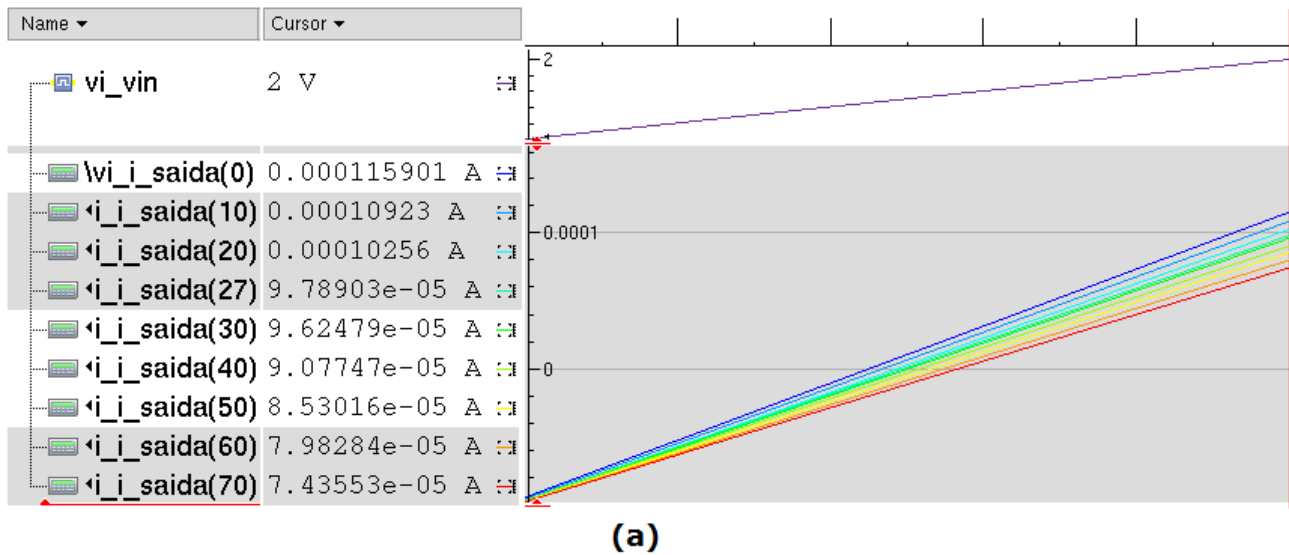
Figura 5.10 – efeito da temperatura no modelo comportamental do V-I:
(a) ao longo da faixa de entrada; (b) nos diferentes modos de operação

5.1.2.3 – Modelo estrutural

Os sub-blocos do modelo estrutural foram modelados e simulados (os código e *testbenches* são dados no Apêndice E) e posteriormente interconectados para implementar o superbloco. Os resultados são apresentados brevemente a seguir.

A Figura 5.11 a seguir mostra o resultado de simulação do núcleo de conversão (entidade

vi_nuc). Pode-se perceber que a saída do núcleo (onda de cima) fica entre $2{,}5.10^{-06}$ e $12{,}5.10^{-06}$, e de forma correspondente à entrada.
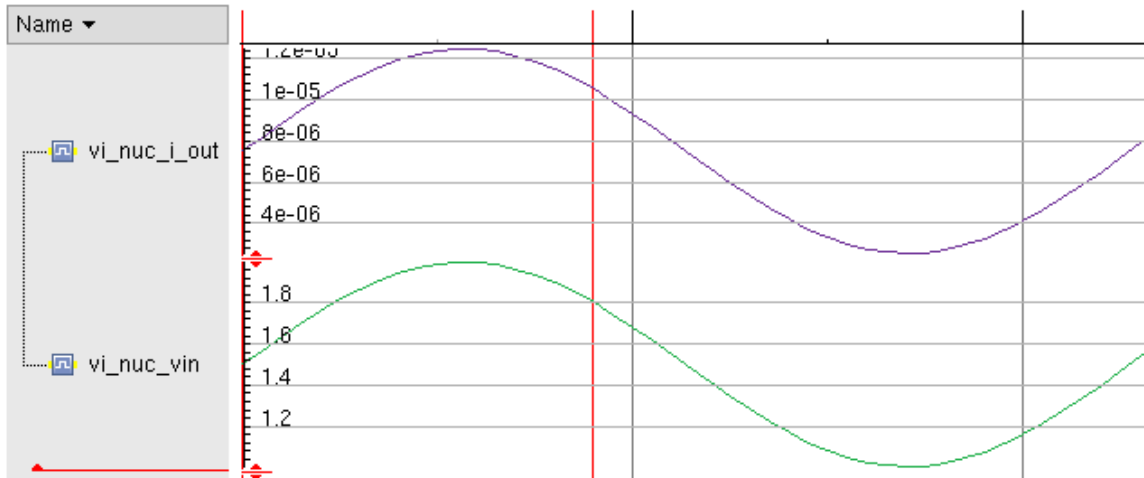
Figura 5.11 – simulação do sub-bloco *vi_nuc*

A referência de corrente simplesmente fornece valores constantes de corrente. Sua simulação é mostrada na Figura 5.12.
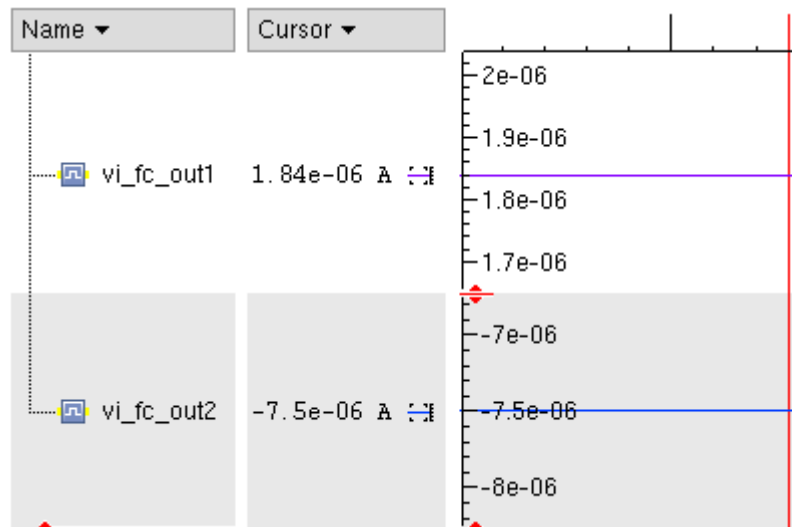
Figura 5.12 – simulação do sub-bloco *vi_ref*

Os sub-blocos vi_g4 e vi_g5 têm implementação muito semelhante. Na Figura 5.13 são mostrados resultados de simulação de (a) *vi_g4* e (b) *vi_g5*.
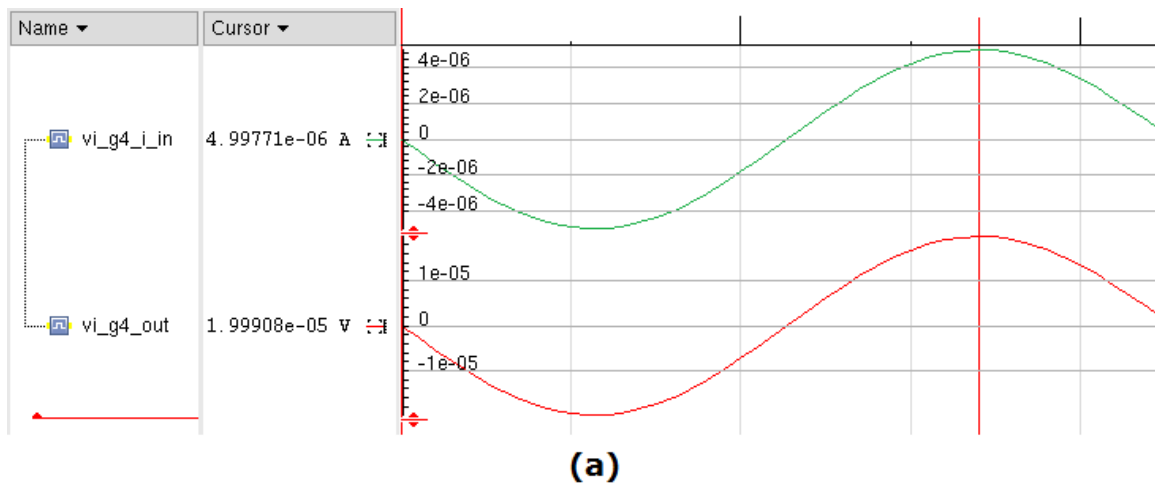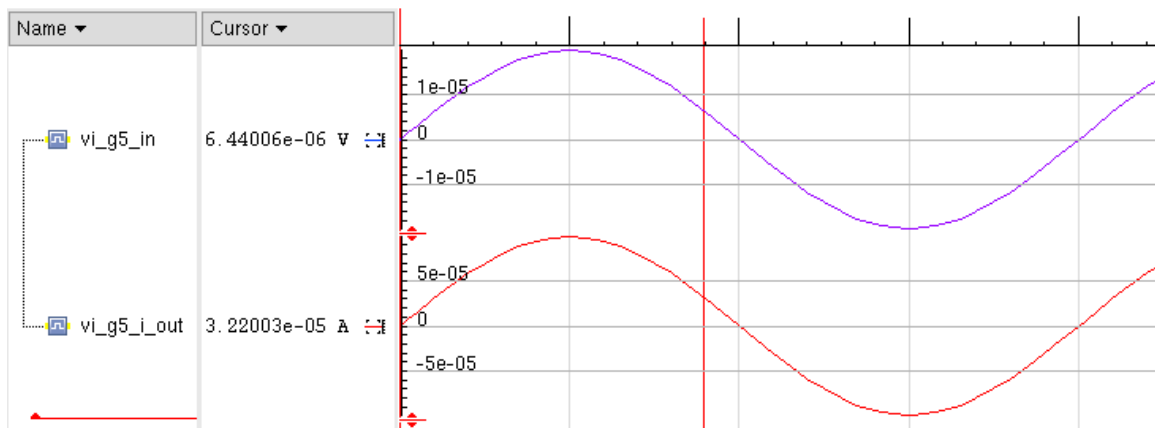
Figura 5.13 – simulação dos sub-blocos *vi_g4* e *vi_g5*

Os sub-blocos, como visto anteriormente, funcionaram em testes individuais. Ao se fazer as simulações de teste do superbloco, não houve erro de compilação, indicando que os códigos estavam corretamente construídos, porém a ferramenta não foi capaz de produzir resultados numéricos que pudessem ser visualizados em função do tempo, por problemas de convergência e cálculo.

A análise das informações fornecidas pela ferramenta de simulação permitiu concluir que o erro se dava ao realizar a conexão interna no nó entre núcleo, referência de corrente e estágio G4. Após algumas tentativas de solucionar esse problema, chegou-se a uma solução de compromisso em que a referência de corrente é eliminada e o núcleo adaptado para fornecer a corrente na faixa adequada à entrada de G4. Como a referência de corrente não tem ligação com a interface externa do superbloco e seu funcionamento se limita ao núcleo, considerou-se que esta solução, embora não ideal, é suficiente para que o modelo seja satisfatório. A estrutura simplificada do modelo, então, fica representada como na Figura 5.14 (uma variação da Figura 4.4). Será feita uma análise mais

minuciosa, no futuro, para a implementação do superbloco como originalmente planejado.
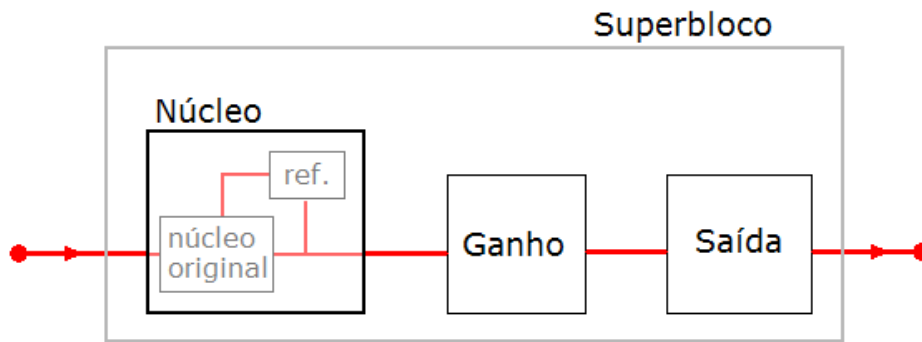
Figura 5.14 – representação do modelo estrutural simplificado do V-I

Com o superbloco representado na Figura 5.14, obtêm-se os resultados de acordo com as especificações do V-I, como mostrado na Figura 5.15. De cima para baixo, as ondas representam a entrada de tensão, a saída do núcleo simplificado, a saída de G4 e a saída de corrente do superbloco. Apesar de as ondas parecerem muito semelhantes, note-se as escalas diferentes (os valores indicados são para o instante indicado em vermelho).
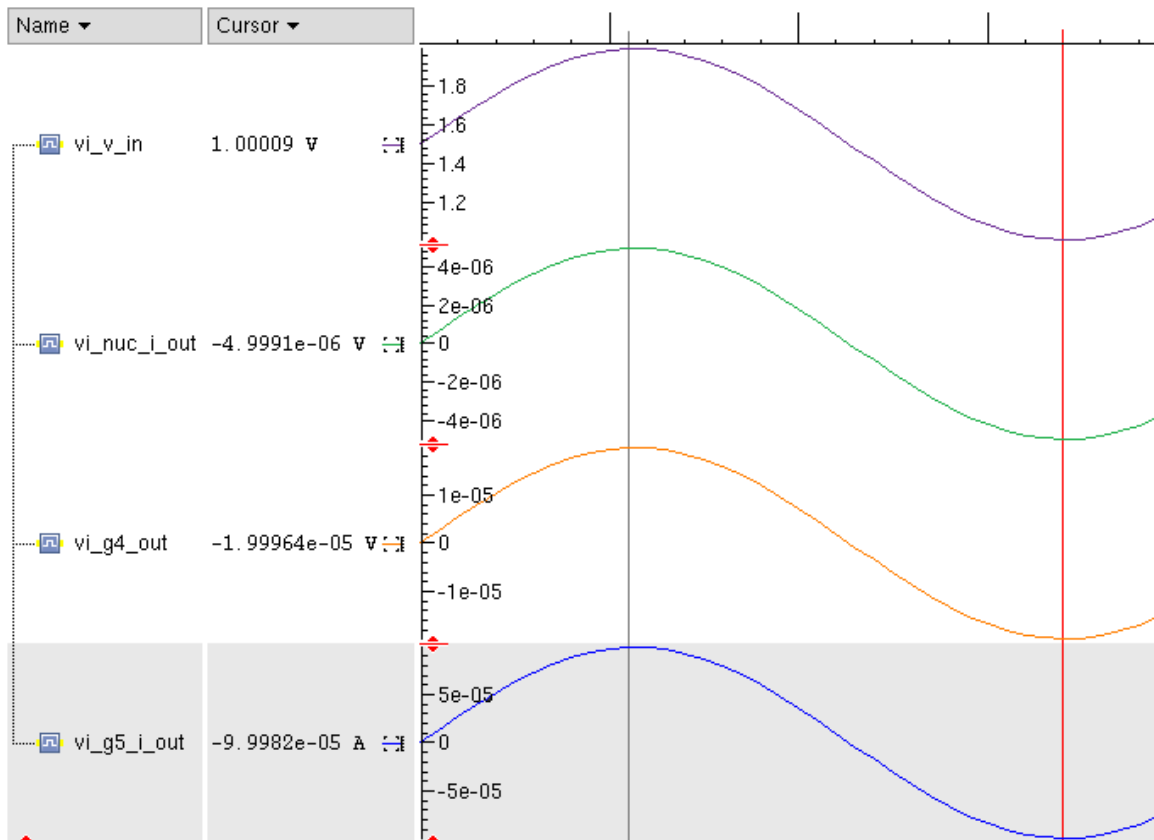
Figura 5.15 – simulação do superbloco do modelo estrutural do V-I

5.2 – CONVERSOR A/D

O conversor A/D, como explicado na documentação, já inclui estruturas de teste, observabilidade, etc. Portanto, o ponto principal da sua aplicação à metodologia são os modelos de alto nível de abstração, descritos a seguir. As alterações de nomenclatura, para adaptar-se à proposta deste trabalho, são citadas no item 5.2.2.

5.2.1 – Modelagem

Nos itens a seguir, serão mostrados os resultados obtidos nas simulações dos modelos do A/D, de acordo com o nível de abstração.

5.2.1.1 – Modelo funcional

Na Figura 5.16 é mostrada parte do resultado de simulação do modelo funcional do A/D. Essa simulação é uma aproximação da interface analógica do SCI; foi realizada interconectando-se, em uma plataforma de teste VHDL-AMS, modelos funcionais de três módulos AMS – um bloco gerando um sinal analógico de tensão, o conversor V-I e o conversor A/D. Na Figura 5.17 é mostrada uma representação simplificada da simulação.
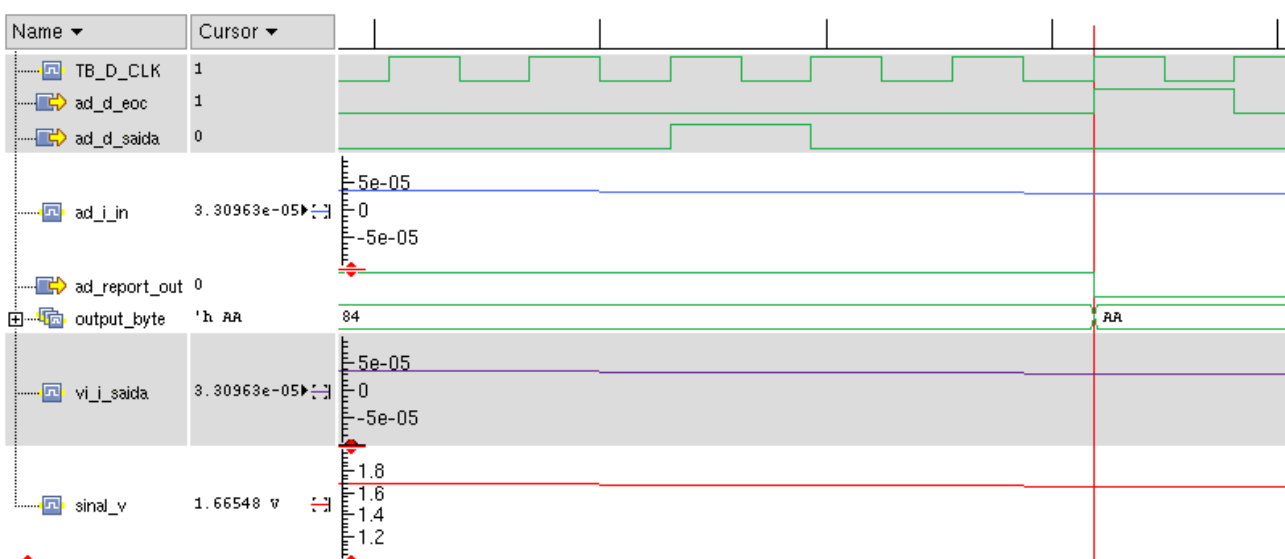
Figura 5.16 – conversão de sinal em modelos ideais da interface analógica

No instante indicado na Figura 5.16 pelo cursor vertical, em vermelho, a seguinte situação pode ser observada: o sinal de tensão é de 1,66548 V. A saída do V-I é de $33{,}0963.10^{-6}$, correspondendo às especificações ideais. A variável `output_byte`, representada em base hexadecimal, assume o valor `AA` – o que corresponde à palavra binária 1010 1010 (ou 170, em base decimal). Esses valores estão de acordo com as especificações ideais do A/D, como se pode verificar pela relação abaixo, onde 169,7 foi aproximado por 170 (já que só se admitem valores inteiros):

$$\left(\frac{(33{,}0963+100)}{200}*255\right)_b=(170)_b=10101010 \qquad (5.1)$$
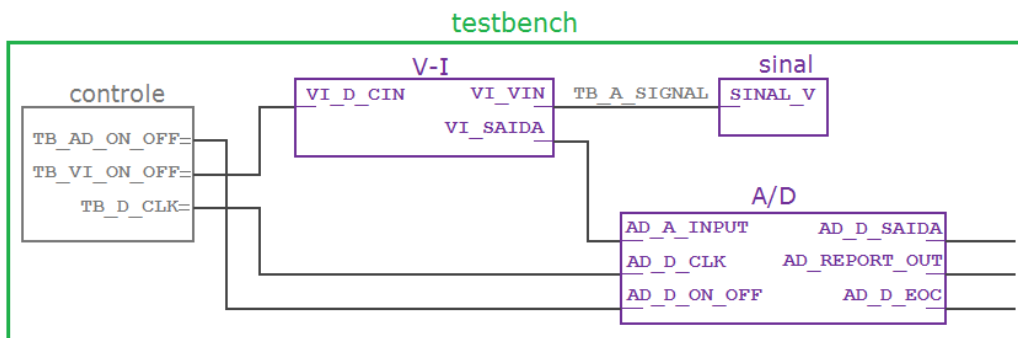
Figura 5.17 – esquemático da simulação VHDL-AMS da interface ideal

Outras características do A/D presentes no modelo funcional são ilustradas na Figura 5.18, obtida da mesma simulação, onde o V-I está desligado (o sinal de tensão varia, mas a corrente de saída do V-I é nula):

- Durante os 8 períodos de relógio anteriores ao cursor em vermelho:
  - o sinal `ad_report_out` assume valor '1';
  - lendo-se o valor do sinal `ad_d_saida`, um bit a cada período de relógio, tem-se o valor `0010 1100` – que corresponde ao número hexadecimal `2C`, armazenado na variável `output_byte`
- No instante em que `ad_report_out` volta para nível baixo:
  - o sinal `ad_d_eoc` sobe, e permanece em '1' durante um período de relógio;
  - um novo valor já está disponível em `output_byte`, pois o resultado da conversão, na simulação, é obtido instantaneamente;
  - o resultado da nova conversão
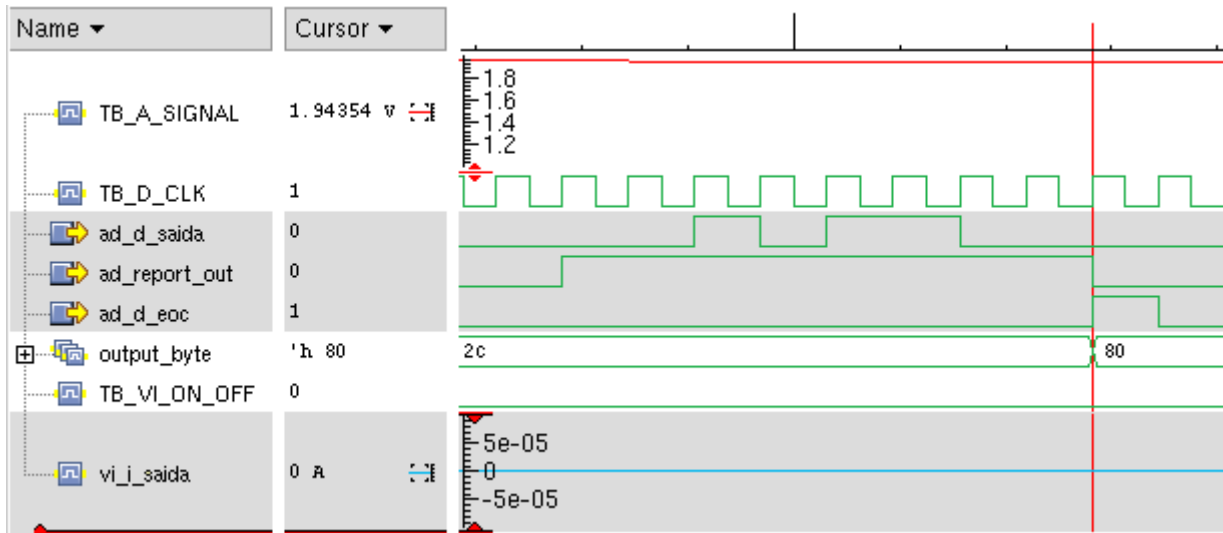
Figura 5.18 – trecho de simulação VHDL-AMS da interface analógica ideal

5.2.1.2 – Modelo comportamental

O comportamento não-ideal do conversor A/D foi modelado de acordo com os procedimentos descritos na seção 4.2.1. Na Figura 5.19, é representado o resultado de conversão analógico-digital obtido em simulação do modelo VHDL-AMS para uma entrada rampa com inclinação constante em toda a faixa de entrada do A/D. O modelo utilizado é comportamental, mas com parâmetros de não-idealidade iguais a zero.

A imagem da saída foi dividida em duas partes para melhor visualização, onde o trecho acinzentado na parte (a) mostra o início da parte (b), e o trecho acinzentado em (b) mostra o final de (a). Todos os valores digitais são obtidos, em ordem crescente. A curva inferior, em cada parte, representa o LSB; a superior representa o LSB.
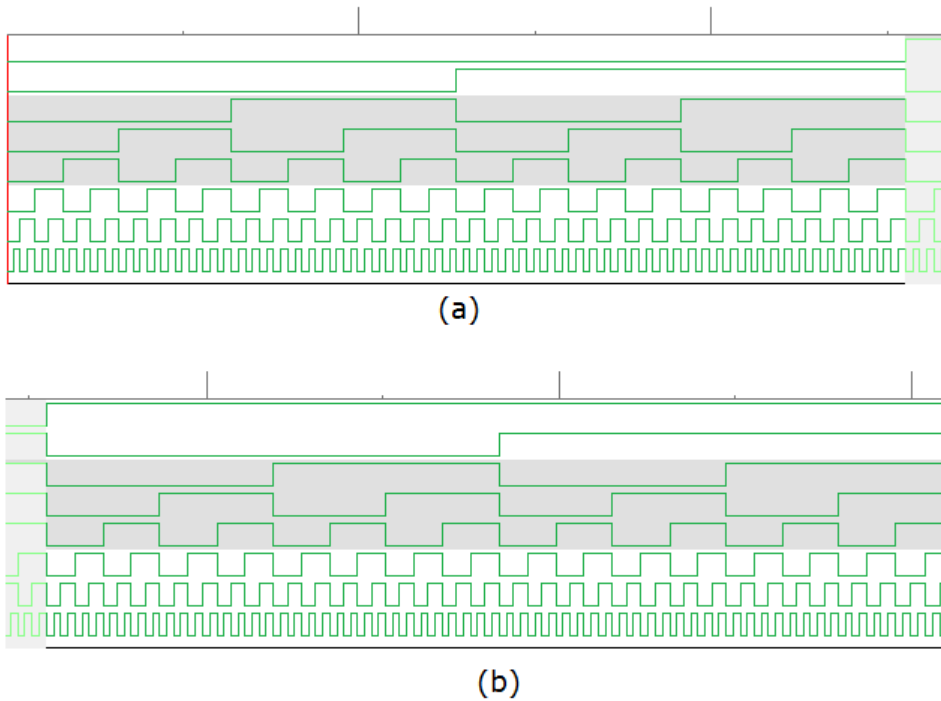
Figura 5.19 – saída digital ideal do A/D, para entrada variando em toda a faixa

(a) 0000 0000 a 0111 1111

(b) 1000 0000 a 1111 1111

Na Figura 5.20, é mostrada uma simulação semelhante, dividida da mesma maneira, mas em que todos os quatro parâmetros tem valores não-nulos:

- offset = 1 LSB
- erro de ganho = 1 LSB
- INL = -1 LSB, com `ad_INL_vlow` = 3,13 μA e `ad_INL_vhigh` = 4,69 μA
- DNL de 1 LSB na saída 1000_0101

Algumas conseqüências facilmente perceptíveis destes efeitos não-ideais, na Figura 5.15 são que a saída não inicia em 0000 0000 e que as saídas 0111 1111 e 1000 0000 não ocorrem.
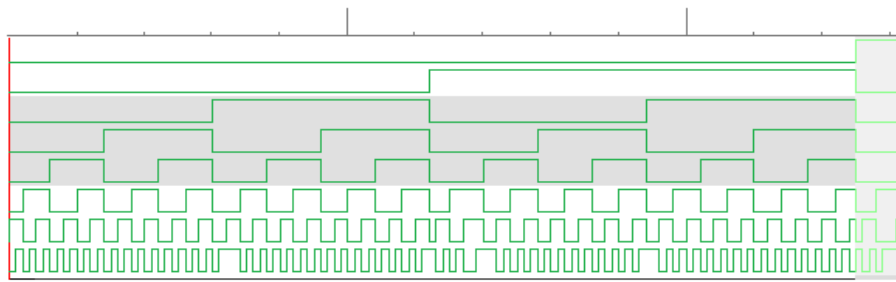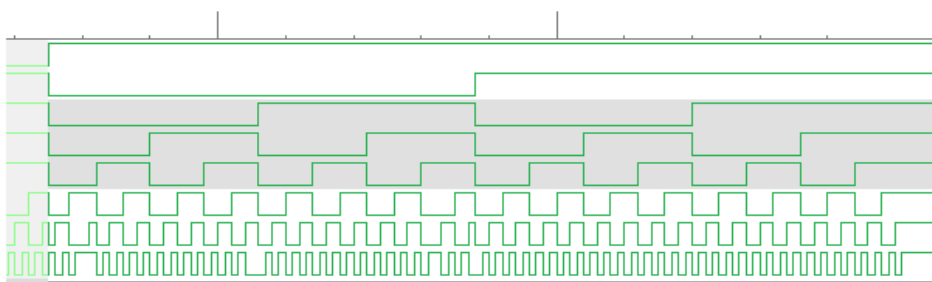
Figura 5.20 – saída digital não-ideal do A/D, para entrada variando em toda a faixa

(a) 0000 0010 a 0111 1110

(b) 1000 0001 a 1111 1111

Como mencionado anteriormente, para avaliar os efeitos não-ideais, é mais vantajoso obter-se curvas entrada/saída. As Figuras 5.21 a 5.24 contêm os gráficos obtidos por simulação dos algoritmos de amostragem, conversão e saída do modelo comportamental, descritos em MatLab. Os códigos são apresentados no Apêndice E. O eixo horizontal de cada figura representa a entrada do A/D, em μA, e o eixo vertical representa a saída binária. Com exceção da Figura 5.22, a visualização foi ajustada para apenas uma faixa da curva, para permitir melhor compreensão do gráfico.

Pode-se perceber que os efeitos não-ideais parametrizados no algoritmo correspondem ao descrito na seção 4.2.1 em todos os casos. As simulações que geraram as curvas nas Figuras 5.21 a 5.24 foram feitas com 8192 pontos para a faixa de entrada.
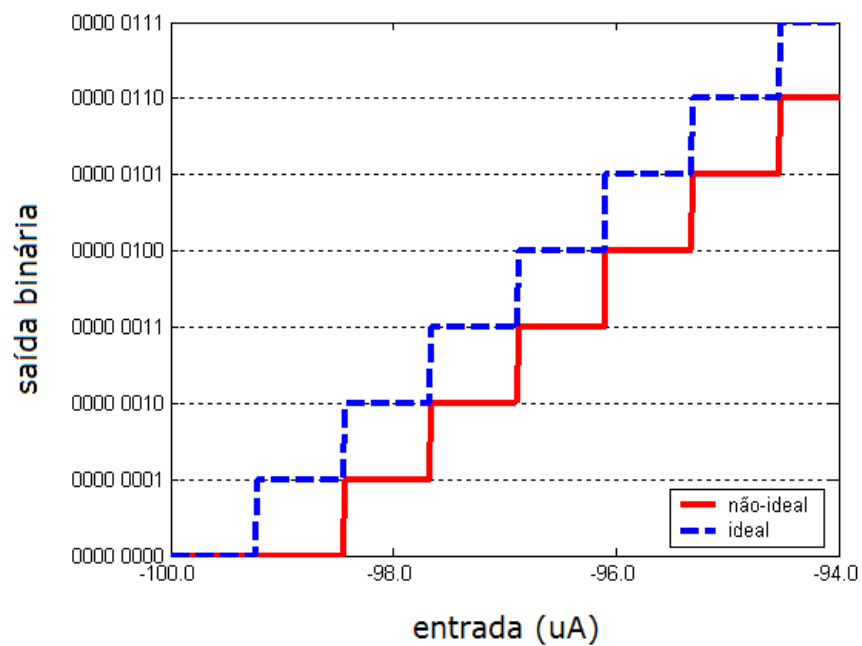
Figura 5.21 – efeitos não-ideais no conversor A/D – offset
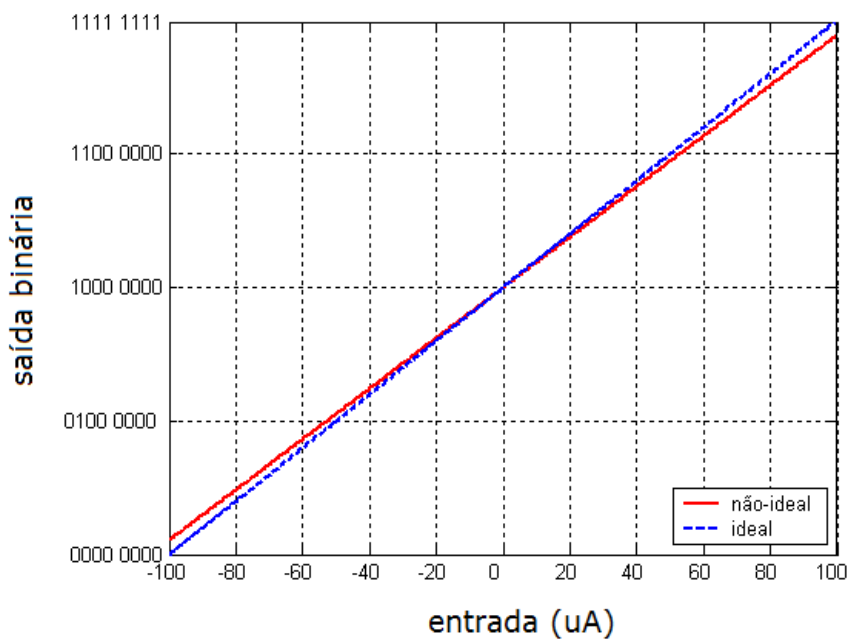
Figura 5.22 – efeitos não-ideais no conversor A/D – erro de ganho
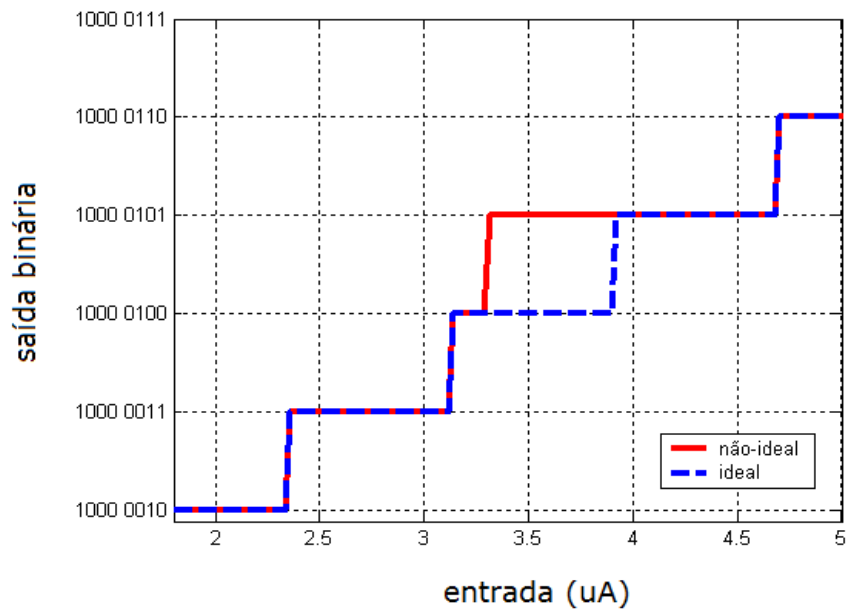
Figura 5.23 – efeitos não-ideais no conversor A/D – INL
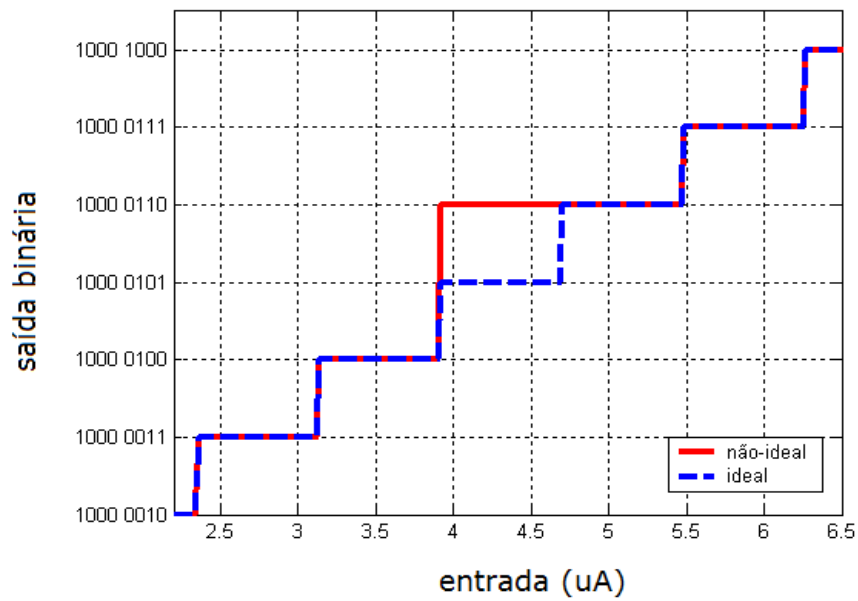
Figura 5.24 – efeitos não-ideais no conversor A/D – DNL

5.2.1.3 – Modelo estrutural

Serão apresentados aqui, primeiramente, os resultados obtidos com os sub-blocos. O sub-bloco *S/H* tem comportamento bastante simples, ilustrado na Figura 5.25. No instante A, recebe o sinal de fim de conversão (`ad_sh_eoc = '1'`), e amostra a corrente; esta é copiada para a saída,

onde permanece constante. No instante C, o *S/H* é desligado (`ad_sh_on_off = '0'`) e a saída do sub-bloco vai a 0 A. Quando é ligado novamente, a corrente é amostrada mais uma vez, ilustrando que os instantes de amostragem são controlados tanto por `ad_sh_eoc` quanto por `ad_sh_on_off`. Isso está de acordo com o código do modelo, que indica o final do ciclo com o comando

```
wait until ad_sh_on_off = '0' or ad_sh_eoc = '1';
```
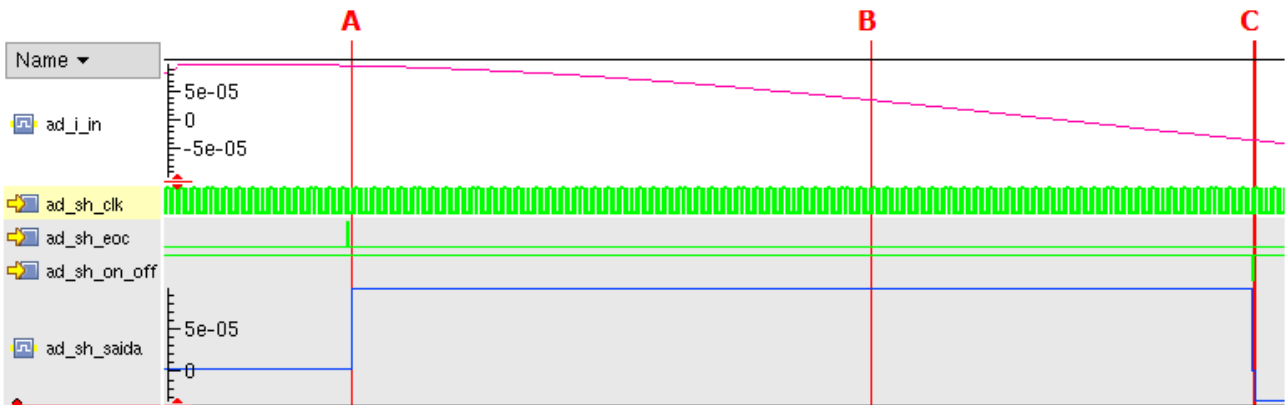
Figura 5.25 – modelo VHDL-AMS do sub-bloco "S/H" em funcionamento

O funcionamento do bloco *Memo* é verificado na Figura 5.26, abaixo.
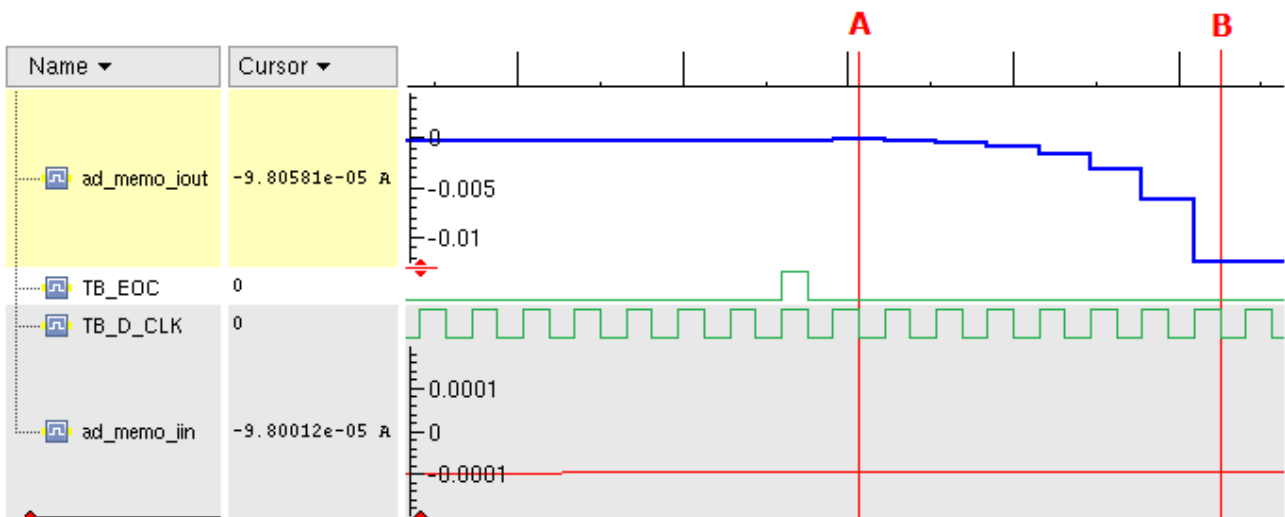
Figura 5.26 – simulação do sub-bloco "Memo"

Um período de relógio após receber o sinal de fim da conversão, a corrente é $-98{,}058.10^{-6}$ A; esta corrente é amostrada e permanece fixa na saída (`ad_memo_iout`) durante um período de relógio (como mostrado pelo instante A, em que a corrente de entrada mudou, mas permanece com

o valor amostrado). Realimentando-se a corrente da saída na entrada bo bloco, a cada período de relógio subseqüente o valor da corrente é dobrado, chegando-se, ao cabo de 7 períodos, à corrente de $12{,}551.10^{-3}$, igual a $128.(-98{,}058.10^{-6})$, mostrada no instante B.

O sub-bloco *Ref* tem seu funcionamento ilustrado na Figura 5.27. Um período de relógio após o sinal de fim da conversão (i.e., no instante "1"), o conversor realiza a primeira comparação – portanto, a saída de corrente de *Ref* (`ad_ref_iout`) é nula. A partir do instante 2, a saída depende do resultado da comparação (lido no sub-bloco pelo bit de entrada `ad_ref_comp`) do *relógio anterior*; caso seja '1', a corrente é +100 µA, do contrário é -100 µA.

Esse comportamento ficará mais claro com o auxílio da Tabela 5.5, onde *t* são os instantes mostrados na Figura 5.27 e *ad_ref_comp (t-1)* é o resultado da conversão no período anterior.
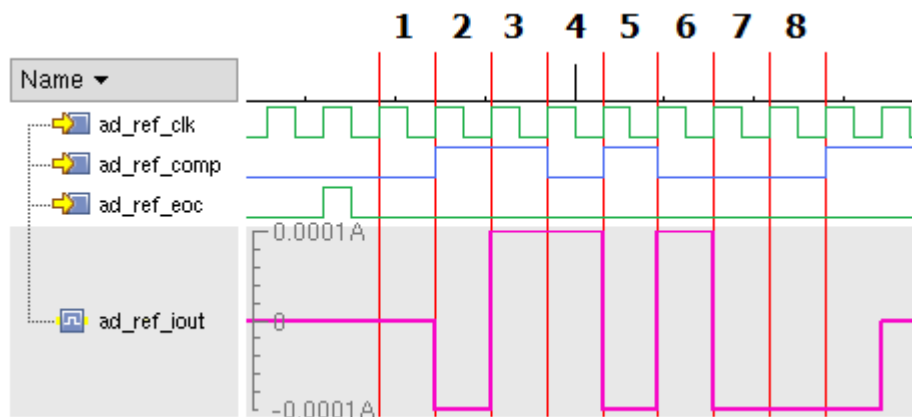
Figura 5.27 – simulação do sub-bloco "Ref"

Tabela 5.5 – resposta esperada e obtida para o sub-bloco "Ref"

t	`ad_ref_comp` (*t-1*)	`ad_ref_iout` esperado (µA)	`ad_ref_comp`	`ad_ref_iout` obtido (µA)
1	-	0	0	0
2	0	–100	1	–100
3	1	+100	1	+100
4	1	+100	0	+100
5	0	–100	1	–100
6	1	+100	0	+100
7	0	–100	0	–100
8	0	–100	0	–100

Pode-se perceber que, da maneira como o sub-bloco foi implementado, o último resultado de conversão é considerado pelo modelo (que responde após o instante 8), porém isso não é necessário para o funcionamento do A/D, pois o último bit já foi convertido. Como mostrado, o sub-bloco *Ref* satisfaz as exigências do A/D.

O modelo do comparador (sub-bloco *Comp*) tem seu comportamento exemplificado na Figura 5.28, abaixo, cujos resultados são listados (inclusive as medidas da entrada nos três instantes mostrados) em seguida na Tabela 5.6. Pode-se perceber o funcionamento do bloco conforme previsto: o bit `ad_report_out` vai a '1' nos instantes em que a comparação é feita (após `ad_comp_on_off = '1'` ou após fim de conversão, indicada pelo sinal `ad_comp_eoc`), e a saída é coerente com o esperado.
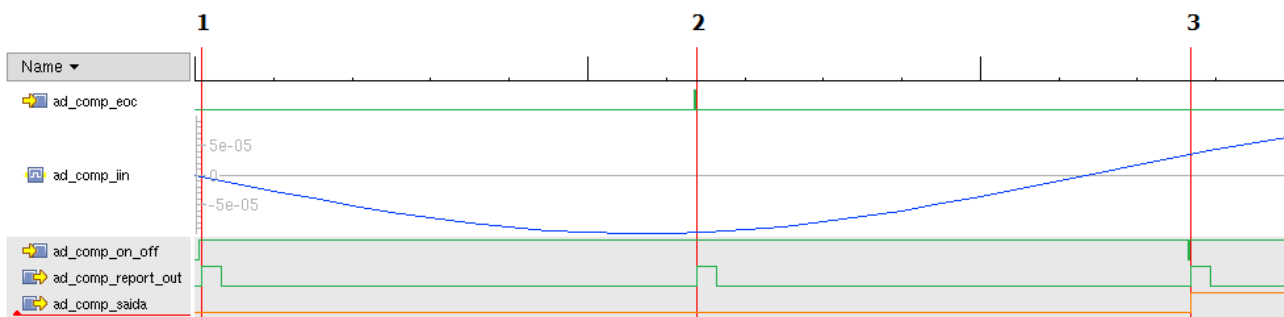
Figura 5.28 – simulação do sub-bloco “Comp”

Tabela 5.6 – resultados esperados e obtidos pelo sub-bloco “Comp”

Instante	Corrente (µA)	Saída obtida	Saída esperada
1	-2,153	0	0
2	-97,93	0	0
3	35,31	1	1

Para ilustrar o funcionamento do último sub-bloco, *Saída*, a forma de onda apresentada na Figura 5.29 omite grande parte do tempo de simulação. Isso porque a função principal de *Saída*, como descrito na seção 4.2.1, é simplesmente adquirir os bits na entrada (`ad_saida_in`) e repeti-los na saída (`ad_saida_out_bit`), após o tempo adequado; os outros sinais de saída são os indicadores de que os bits estão sendo entregues na saída do bloco (`ad_saida_report_out`) e o sinal de fim de conversão (`ad_saida_eoc`), que só são ativos no final do ciclo de conversão, então a maior parte da simulação deste sub-bloco é ociosa.
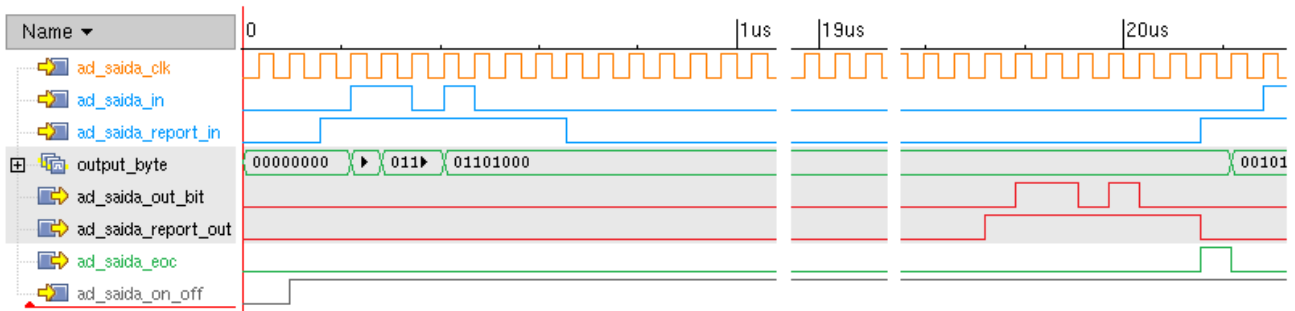
Figura 5.29 – simulação do sub-bloco "Saída"

Quanto à implementação de um superbloco instanciando os quatro sub-blocos, houve dificuldades também no modelo estrutural do A/D. Devido a problemas com a interface do bloco *Memo*, foi criado um bloco auxiliar, *ad_sum*, para selecionar a corrente de entrada de *ad_memo*. Como se pode perceber pelos resultados da simulação apresentados na Figura 5.30, o superbloco realiza a amostragem da corrente e a conversão para 8 bits; ainda assim, porém, o valor resultante da conversão não é o esperado.
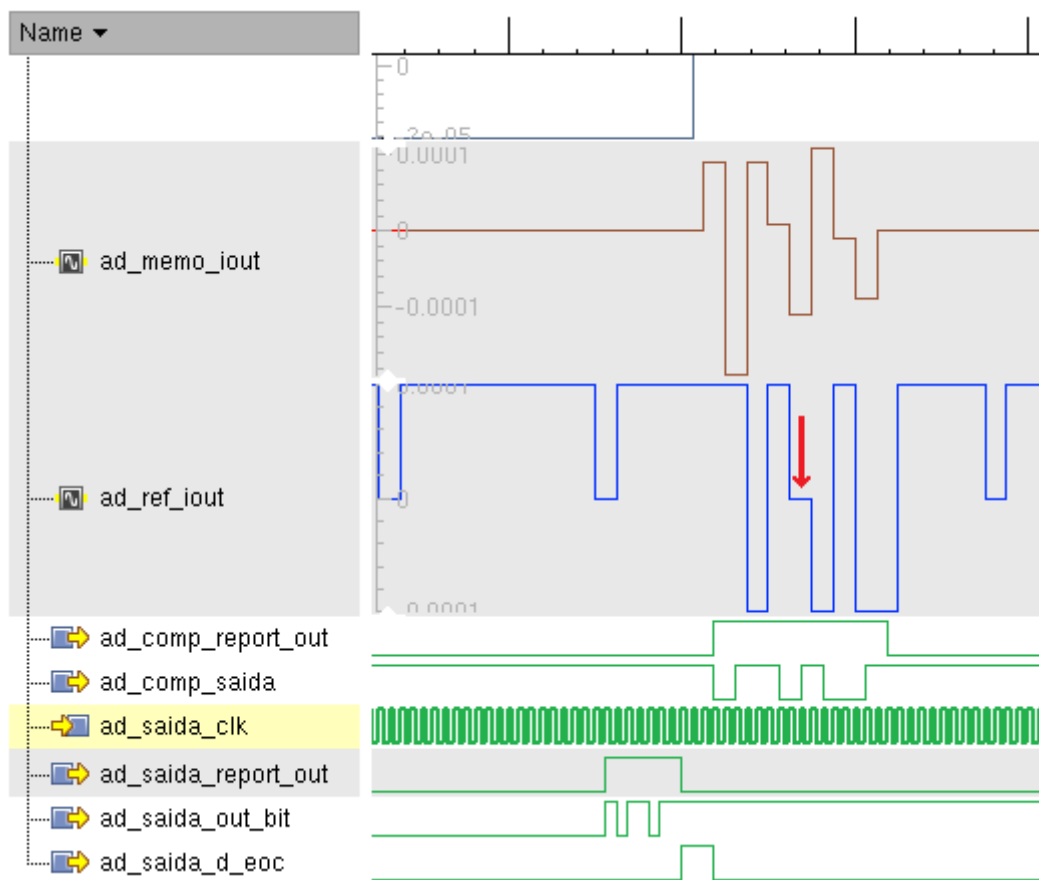
Figura 5.30 – simulação do superbloco do modelo VHDL-AMS do A/D

A Figura 5.30 mostra um período de tempo em que uma palavra é entregue na saída e outra palavra é amostrada e convertida pelo comparado. O resultado da conversão é coerente com as correntes resultantes da soma de *ad_memo_iout* com *ad_ref_iout*, porém fica evidente que há erros na funcionalidade verificando-se que, no meio dos 8 ciclos de conversão, a referência de corrente assume valor 0 (indicado pela seta vermelha, na Figura).

Como pode ser visto nos códigos no Apêndice E, tentou-se sincronizar o funcionamento do bloco através dos sinais de relógio, fornecidos a todos os sub-blocos. Em simulação, percebeu-se que quantidades analógicas eram atualizadas no instante seguinte a sinais digitais. Por exemplo, se no instante *t* o relógio vai a '1' e isso causa alterações em terminais e em sinais digitais, os sinais digitais já têm seu novo valor em *t*, enquanto as quantidades analógicas são atualizadas apenas no instante imediatamente posterior (t^+). Pelos resultados obtidos, é provável que o problema no funcionamento do superbloco esteja na sincronia entre os sub-blocos. Por isso, começou a ser desenvolvido uma versão deste modelo que não dependa do relógio exceto para a saída serial, utilizando sinais entre os sub-blocos para coordenar o processo de conversão. Espera-se que futuramente essa alternativa permita implementar o modelo estrutural completo do A/D com sucesso.

5.2.2 – Adaptação da nomenclatura

Os nomes de pinos e sinais do A/D foram adaptados, quando necessário, para serem compostos do prefixo *ad*, seguido, opcionalmente, por uma letra indicando tipo de sinal ou pino, seguida por um nome indicando sua função no bloco. O tipo de sinal ou pino foi indicado por:

- *d* – pino digital (no leiaute) e sinal ou porta digital (nos modelos);
- *a* – pino analógico (no leiaute) e terminal analógico (nos modelos);
- *i* – sinal de corrente;
- *v* – sinal de tensão.

Para a maioria dos pinos, essa adaptação consistiu somente da adição do prefixo, do tipo de sinal e da representação em letras minúsculas. Exceções são os seguintes pinos:

- *AD_D_RD_Out_Byte* passou a ser chamado *ad_d_saida*;
- *ConvDone* passou a ser chamado *ad_d_eoc*.

As demais alterações são apresentadas no Apêndice D.

5.2.3 – Identificação

A etiqueta inserida no leiaute do conversor A/D segue o padrão utilizado na identificação do V-I. É mostrada na Figura 5.31, a seguir.
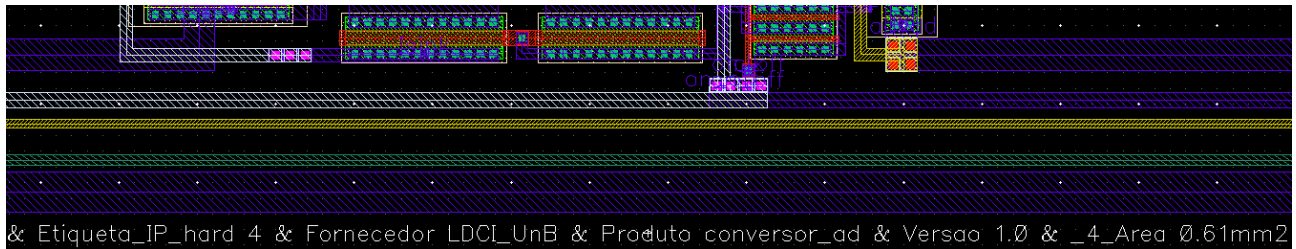
Figura 5.31 – etiqueta de IP no leiaute do A/D

6 – CONCLUSÕES

Foi proposta uma metodologia de adaptação de circuitos VLSI analógicos e de sinal misto para a elaboração de blocos de IP. A aplicação da metodologia a dois blocos previamente projetados obteve resultados consistentes e coerentes com a proposta. No caso do conversor V-I, especialmente, alterações feitas ao projeto original garantiram que o circuito apresentasse características adequadas a um componente virtual AMS, conforme avaliado por meio de pesquisa bibliográfica.

Os modelos de alto nível, validados por simulação, mostraram resultados satisfatórios, compatíveis com os blocos originais. Com tal modelagem aliada à documentação elaborada, é possível avaliar o funcionamento do circuito e a sua possibilidade de integração sem conhecer detalhes da topologia interna ou do dimensionamento do circuito, portanto preservando a propriedade intelectual do criador.

Em trabalhos futuros, poderá ser estudada mais aprofundadamente a implementação de um desligamento completo do conversor V-I (isolando-o da alimentação elétrica), para avaliar a possibilidade de se evitar uma degradação impeditiva do desempenho, devido ao inevitável redimensionamento do circuito. Outras perspectivas de aprimoração futura do trabalho são a criação e otimização de variações dos conversores, para criar efetivamente uma biblioteca de células com

certa flexibilidade de especificações. Isso pode ser feito a partir de uma parametrização mais detalhada dos circuitos, que facilite a migração de tecnologias de fabricação.

A modelagem dos blocos em outras linguagens de descrição de hardware AMS já está sendo feita (primeiramente, criando-se modelos do A/D em Verilog-AMS), e espera-se utilizar os modelos construídos em co-simulações com os modelos de alto nível das seções digitais do SoC (o que pode ser útil também para avaliar as vantagens e desvantagens de cada linguagem [55]). Os modelos estruturais, em especial, serão melhorados. Após a caracterização dos protótipos dos blocos, a informação obtida poderá ser usada para refinar também o modelo comportamental.

As aplicações futuras descritas, e a possibilidade de realizá-las, são uma continuação do trabalho apresentado aqui. Feitas essas considerações, considerou-se que os objetivos propostos foram atingidos, embora ainda haja espaço para aperfeiçoamento da metodologia proposta, especialmente após a avaliação desta primeira aplicação.

REFERÊNCIAS BIBLIOGRÁFICAS

[1] Allen, P. E. e Holberg, D. R., *CMOS Analog Circuit Design*, 2ª Ed., Oxford University Press, EUA, 2002.

[2] Gielen, G.G.E. e Rutenbar, R.A. *Computer-Aided Design of Analog and Mixed-Signal Integrated Circuits.* Proceedings of the IEEE, vol. 88, nº12, 2000.

[3] Araújo, W. A., *Proposta de adaptação de um núcleo de processadores RISC 16 bits CMOS ao padrão VSIA para propriedade intelectual de semicondutor*, projeto final de graduação, Universidade de Brasília, 2006.

[4] Saleh, R., Wilton, S., Mirabbasi, S., Hu, A., Greenstreet, M., Lemieux, G., Pande, P.P., Grecu, C. e Ivanov, A., *System-on-Chip: Reuse and Integration*, Proceedings of the IEEE, vol. 94, nº 6, EUA, 2006.

[5] N. M. Madrid, E. Peralias, A. Acosta, A. Rueda, *Analog/Mixed-Signal IP Modeling for Design Reuse*, Proceedings DATE Conference, pp. 766-767, Alemanha, 2001.

[6] Vandenbussche, J., Gielen, G. e Steyaert, M., *Systematic Design of Analog IP Blocks*, Kluwer Academic Publishers, Países Baixos, 2003.

[7] www.vsi.org

[8] Castro-López, R., Fernández, F. V., e Vázquez, A. R., *A reuse-based framework for the design of analog and mixed-signal ICs*. Proceedings of the SPIE, 5837(3), pp. 25-36, 2005.

[9] Hastings, A., *The Art of Analog Layout*, 2ª Ed., Prentice Hall, EUA, 2005.

[10] VSI Alliance, *Virtual Component Attributes (VCA) With Formats for Profiling, Selection, and Transfer Standard (version 2.3)*, 2003.

[11] Freescale Semiconductor, *Semiconductor Reuse Standard – Documentation (v 3.2)*, 2005.

[12] Castro-López, R., Fernández, F. V., Delgado-Restituto, M., Medeiro, F. e Rodríguez-Vázquez, A., *Creating Flexible Analogue IP Blocks*. Proceedings of the 27th European Solid-State Circuits Conference (ESSCIRC 2001), pp. 437-440, 2001.

[13] Li, Z., Luo, L. e Yuan, J., *A Study on Analog IP Blocks for Mixed-Signal SoC*, Proceedings ASIC, pp.564-567, Beijing, 2003.

[14] Levi, T., Tomas, J., Lewis, N. e Fouillat, P., *IP-Based design reuse for analog systems*, Proc. SPIE, vol. 6590, França, 2007.

[15] Bourguet, V., de Lamarre, L. e Rosset-Louërat, M.M., Analog IC Design with a Library of Parameterized Device Generators, DCIS 2004, França, 2004.

[16] Iskander, R., Louërat, M.M. e Kaiser, A., *Automatic DC Operating Point Computation and Design Plan Generation for Analog IPs*. Analog Integrated Circuits and Signal Processing

Journal, Vol. 56, Issue 1-2, pp. 93-105, 2008.

[17] Iskander, R., Galayko, D., Louërat, M.M. e Kaiser, A., *Knowledge-Aware Synthesis Using Hierarchical Graph-Based Sizing and Biasing*. 50th IEEE International Midwest Symposium on Circuits and Systems (MWSCAS'07), pp. 984-987, Canadá, 2007.

[18] Doménech-Asensi, G., Ruiz-Merino, R., Madrid, J.A.D. e Neubauer, H., *Evaluation of VHDL-AMS models of a high performance ADC*. IEEE International Symposium on Industrial Electronics (ISIE 2007), Espanha, 2007.

[19] *Motorola Announces Intellectual Property Interface (IPI) Standard To Speed Development Of Embedded Applications*, D&R Headline News, www.design-reuse.com , 1999.

[20] www.freescale.com

[21] www.spiritconsortium.org

[22] VSI Alliance, *Analog/Mixed-Signal VSI Extension Specification (version 2.2)*, 2001.

[23] Bailey, B. e Werner, K., Intellectual Property for Electronic Systems: An Essential Introduction, International Engineering Consortium, EUA, 2007.

[24] Freescale Semiconductor, *Semiconductor Reuse Standard – VC Block Deliverables (v 3.1.1)*, 2003.

[25] Hamour, M., Saleh, R., Mirabbasi, S. e Ivanov, A., *Analog IP Design Flow for SoC Applications*. University of British Columbia, Canadá, 2003.

[26] Daniel Gajski, D. e Kuhn, R.H., New VLSI Tools. IEEE Computer 16(12), pp. 11-14, EUA, 1983.

[27] Ashenden, P. J., Peterson, G. D. e Teegarden, D. A., *The System Designer's Guide to VHDL-AMS*, Morgan Kaufman, EUA, 2003.

[28] *Verilog-AMS Language Reference Manual – Analog & Mixed-Signal Extensions to Verilog HDL (Version 2.3)*, Accellera Organization, 2008.

[29] IEEE, *IEEE Standard VHDL Analog and Mixed-Signal Extensions*, IEEE Std 1076.1-2007, 2007.

[30] Pêcheux, F., Lallement, C. e Vachoux, A., *VHDL-AMS and Verilog-AMS as Alternative Hardware Description Languages for Efficient Modeling of Mutidiscipline Systems*. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 24, nº 2, EUA, 2005.

[31] Cadence Design Systems, *NC-VHDL Simulator Help (Version 8.1)*, 2008.

[32] Evans, J., *High-level modelling tools*. Uppsala Universitet, Suécia, 2007.

[33] www.mentor.com

[34] www.synopsis.com

[35] Cadence Design Systems, *SimVision User Guide (Version 8.1)*, 2008.

[36] Costa, J.C., Rocha, A.F., Menezes, L.R.A.X., Jacobi, R.P., Romariz, A.R.S., Soares, R.R.P., Beserra, G.S., Costa, J.D., Araújo, G.M, Araújo, W.A., Marra, J.C.S.S., Amaral, W.A., Vogel, P.R.O., da Silva, A.L., Martins, A.J .O. e Povoa, L.R., *CMOS SoC for Irrigation Control*, Proceedings of the 2005 SOCC (IEEE International SOC Conference), EUA, IEEE Press, pp. 51-54, 2005.

[37] Araújo, G.M., *Conversor Tensão-Corrente em Tecnologia CMOS para um Conversor Analógico/Digital de um Sistema em Chip*, Dissertação de Mestrado, Universidade de Brasília, Brasil, 2008.

[38] Pimentel, J. V. B., *Interface de aquisição de dados analógicos para sistema em chip utilizando sensores externos*, projeto final de graduação, Universidade de Brasília, 2006.

[39] Soares, V.F., Medeiros, J.E.G. e Costa, J.C., *CMOS A/D Converter for SOC in Wireless Sensor Network Applications*, aceito para publicação em 8º Seminário Internacional de Metrologia Elétrica (VIII SEMETRO), Brasil, 2009.

[40] Soares, V.F., *Projeto de Conversor Analógico-Digital Cíclico para Sistema em Chip CMOS*, Relatório de iniciação científica, Universidade de Brasília, Brasil, 2008.

[41] *Documentação de projeto EPUSP_NAMITEC*, LDCI, Universidade de Brasília, 2007-2008.

[42] Araújo, G.M., Madureira, H.M. e Costa, J.C., *Design and characterization of a 0.35 um CMOS Voltage-to-current converter.* Aceito para publicação em 22nd Symposium on Integrated Circuits and Systems Design (SBCCI 2009), Brasil, 2009.

[43] Looby, C.A. e Lynden, C., *Field Programmable Analogue Arrays: a DFT View*. IEE Colloquium on Testing Mixed Signal Circuits and Systems, 1997.

[44] VSI Alliance, *Virtual Component Transfer Specification (version 2.1)*, 2001

[45] Freescale Semiconductor, *Semiconductor Reuse Standard – Functional Verification (v 3.1.1)*, 2003.

[46] VSI Alliance, *Soft and Hard VC Structural, Performance and Physical Modeling Specification (version 2.1)*, 2001.

[47] VSI Alliance, *Virtual Component Identification Soft IP Tagging Standard (version 2.0)*, 2006.

[48] VSI Alliance, *Virtual Component Identification Physical IP Tagging Standard (version 3.0)*, 2006.

[49] Abt, J., Gont, V., Zavadsky, V. e Choi, Y., *Identifying IP cores – to protect your investment*. Design & Reuse (www.design-reuse.com), Canadá, 2009.

[50] Open SystemC Initiative (OSCI), *Draft Standard SystemC AMS Extensions – Language Reference Manual*, 2008.

[51] Cadence Design Systems, *Virtuoso AMS Designer Simulator User Guide (Version 8.2)*, 2008.

[52] http://www.bipm.org/en/si/

[53] Razavi, B., *Design of Analog CMOS Integrated Circuits*, McGraw-Hill, Cingapura, 2001.

[54] Austria Microsystems, *0.35 μm CMOS C35 Process Parameters (Document Number: ENG-182) Rev. 2.0*, 2003.

[55] Pimentel, J.V.B. e Costa, J.C., *VHDL-AMS Modeling of Analog/Mixed-Signal IP Blocks*. Aceito para publicação em 9th Microelectronics Students (SFORUM 2009), Brasil, 2009.

APÊNDICES

APÊNDICE A – ELEMENTOS DE SINTAXE EM VHDL-AMS

Serão explicadas aqui algumas características da linguagem VHDL-AMS, complementando o conteúdo apresentado anteriormente. O objetivo é apenas o de fornecer uma referência rápida para facilitar a compreensão dos códigos gerados, apresentados no Apêndice E, não tendo a intenção de ser uma explicação completa ou abrangente da linguagem.

A.1 – OBJETOS E IDENTIFICADORES

A estrutura geral para declaração de um *objeto* é:

```
<classe> <identificador> : <tipo ou natureza> ;
```

O nome do objeto é definido, pelo projetista, por um *identificador*. Identificadores são utilizados não só em objetos, mas também para dar nomes únicos a processos, declarações, etc. - nestes casos são chamados de *etiquetas*. Identificadores devem seguir algumas regras:

- Podem incluir apenas caracteres alfanuméricos ("A" a "Z", "a" a "z", "0" a "9") e traço inferior ("_", chamado também de traço baixo);
- Devem começar com uma letra;
- Não podem terminar com traço inferior;
- Não podem conter dois traços inferiores consecutivos.

A capitalização não é relevante, assim o objeto "Adc" é o mesmo que "ADC" ou "adc". Porém, mesmo que sigam as regras acima, ainda há outra restrição: um identificador não pode corresponder a nenhuma das palavras reservadas da linguagem. Uma lista das palavras reservadas em VHDL-AMS é apresentada no Apêndice B, ao lado das palavras reservadas em Verilog-AMS.

A.2 – QUEBRAS DE LINHA

Comandos em geral (não apenas declarações de objetos) são terminados por ";". Não é necessário, entretanto, que comandos sejam dados em linhas diferentes; declarações distintas em uma mesma linha são interpretadas como tal, desde que cada uma seja seguida por ";". No entanto,

quebras de linha no meio de um comando não são válidas, embora haja exceções (como declarações condicionais **`if`**, cuja estrutura abrange mais de uma linha). Por exemplo:

```
IF cte_1 = 2.0 THEN
     var_1 := 1.0 + cte_1;
ELSE
     var_1 := 0.0;
END IF;
```

Nota-se que é necessário incluir “;” após as atribuições de valores a `var_1`, mas não após **`THEN`** ou **`ELSE`**. De fato, as quebras de linha após **`THEN`** e **`ELSE`** não são apenas válidas, mas obrigatórias.

A.3 – CLASSES, TIPOS E NATUREZAS

Existem seis classes de objetos: terminais, quantidades, constantes, variáveis, sinais, e arquivos. Simplificadamente:

- *Terminais* representam nós dos circuitos e serão explicados adiante;
- *Quantidades* são objetos com valores analógicos, que podem ser associados a terminais;
- *Constantes* são objetos cujo valor atribuído não muda;
- *Variáveis* e *sinais* são objetos cujos valores podem ser modificados durante a execução do código;
- *Arquivos* são uma classe especial de objetos destinados a armazenar informação enquanto o código não estiver sendo executado (i.e., mesmo depois de finalizada a simulação).

Cada classe aceita objetos de determinado *tipo* ou, no caso de terminais, *natureza*. *Tipos* e *naturezas* identificam quais valores podem ser atribuídos a determinado objeto, e como esta atribuição deve ser feita. Por exemplo, a um objeto de tipo *integer* podem ser atribuídos apenas valores inteiros; a um objeto de tipo *real*, podem ser atribuídos valores numéricos inteiros ou fracionários, mas é necessário em todo caso que pelo menos uma casa decimal seja declarada. Assim,

```
constant constante_integer1 : integer := 1;
constant constante_real1 : real := 1.0;
```

e

```
constant constante_integer2 : integer := constante_integer1;
```

são declarações válidas, porém

```
constant constante_real2 : real := 1;
constant constante_integer2 : integer := constant_real1;
```

e

```
constant constante_real2 : real := constante_integer1;
```

não são, pois mesmo que os valores numéricos sejam todos inteiros, os tipos dos objetos devem corresponder ao que está sendo atribuído.

A.4 – ATRIBUIÇÃO DE VALORES

Valores são atribuídos a variáveis e constantes por meio de uma declaração da forma:

```
<identificador> := <expressão> ;
```

Quando este comando for executado, o objeto identificado à esquerda do símbolo "**:=**" recebe o valor indicado pela expressão à direita. Essa forma de atribuição de valores pode ser associada à declaração de um objeto, como visto acima, resultando em:

```
<classe> <identificador> : <tipo ou natureza> := <expressão> ;
```

Se o objeto declarado for uma constante, assume o valor atribuído; se for uma variável, sinal ou quantidade, a expressão atribui-lhe um valor inicial. Do contrário, o valor inicial é um valor padrão que depende do tipo do objeto.

A atribuição de valores a *sinais* é diferente daquela feita a variáveis e constantes, e é expressa como:

```
<identificador> <= <expressão> [after <tempo>] ;
```

A principal diferença entre a estrutura acima e as apresentadas anteriormente é a

possibilidade de se incluir o trecho opcional **after** <*tempo*>, que permite que o sinal assuma o valor determinado pela expressão à direita de <= com certo atraso. Isso é chamado de "agendamento de transição"; é possível agendar mais de uma transição (em tempos diferentes) na mesma linha, separados por vírgula, como por exemplo:

```
clk <= '0' after 0.5 ns, '1' after 1.0 ns;
```

O sinal **`clk`** assumirá o valor '0' 0,5 ns após o instante de execução da linha acima, e, depois de mais 0,5 ns (totalizando 1 ns após a execução da linha), assumirá o valor '1'.

As declarações acima são todas seqüenciais; declarações *simultâneas*, definidas no corpo de uma arquitetura, têm a seguinte forma:

```
<identificador> == <expressão> ;
```

O objeto identificado na parte esquerda da declaração acima vai ter o valor da expressão a qualquer instante. Quando quer que o resultado da expressão mudar, o valor do objeto mudará instantaneamente. Por exemplo, a expressão

```
tens == corr*res;
```

significa que o valor da quantidade `tens` será igual ao produto dos valores de `corr` e `res`. Quando um dos dois objetos à direita tiver seu valor alterado, o valor de `tens` será alterado também; se variarem continuamente, `tens` também variará.

A.5 – TERMINAIS E QUANTIDADES

Terminais representam nós físicos em circuitos, podendo corresponder a um dentre vários domínios de energia de um sistema. Por motivos óbvios, o enfoque aqui é dado a terminais de natureza elétrica; a não ser que seja dito em contrário, todos *terminais* tratados neste trabalho representam nós elétricos.

Não é feita atribuição de valores a terminais, porém a eles são associadas *quantidades*. Estas são como variáveis: objetos aos quais são atribuídos valores de determinado tipo, que podem ser alterados durante a execução do modelo. A associação de quantidades a terminais é feita de maneira implícita ou explícita; isso ocorre porque uma *natureza* em VHDL-AMS pode ser considerada como um *tipo* especial, ao qual são associados um tipo *across* (que, por falta de opção melhor, será traduzido como "*ao longo de*"), um tipo *through* ("*através de*") e uma referência. No caso da natureza *elétrica*, a quantidade ao longo do terminal é do tipo *tensão*, a quantidade através do terminal é do tipo *corrente*, e a referência, a não ser que definida pelo projetista, é um terminal padrão denominado *electrical_ref*, que atua como o terminal terra do circuito.

Como um terminal tem, necessariamente, sua natureza determinada, os tipos das quantidades ao longo e através do terminal são também determinados, mesmo que elas não sejam acessadas ou definidas explicitamente. Assim, um terminal de natureza elétrica pode ter as quantidades *across* e *through* definidas e identificadas pelo projetista, ou apenas servir como um nó de conexão do circuito, onde serão ligados outros terminais. Se tal conexão for feita, o modelo interpreta que a relação entre os terminais respeita a leis elétricas (por exemplo: terminais conectados têm mesma tensão).

Também é possível declarar quantidades *de interface* – portas às quais se associa uma quantidade (e não um terminal ou sinal) – e quantidades *livres*, ou seja, objetos analógicos que não são associados a nenhum terminal ou porta. O número de equações simultâneas necessário a uma arquitetura também é definido pelas quantidades declaradas; deve ser igual ao número de quantidades *through*, mais o número de quantidades *livres*, mais o número de quantidades *de interface* de modo *out* (de saída). Caso seja exigido por esta regra que haja mais declarações simultâneas do que de fato necessário para modelar o comportamento do sistema, podem ser usadas quantidades *across* e relacioná-las a quantidades *through*.

A.6 – COMENTÁRIOS

Qualquer conteúdo inserido em um código VHDL-AMS após os caracteres "--" é ignorado pelo simulador ou compilador até o final da linha. Isso é amplamente utilizado em linguagens de programação ou descrição de sistemas para que o projetista possa incluir comentários textuais no código, mas que não serão interpretados como parte dele. Não existe, em VHDL-AMS, estrutura

para “comentários em bloco”, que permita delimitar início e fim do conteúdo a ser ignorado. Isso existe em outras linguagens, como Verilog-AMS, mas em VHDL-AMS o conteúdo ignorado (ou “comentado”) sempre termina no fim da linha.

A.7 – NOTAÇÃO EXPONENCIAL

Pode-se escrever um número seguido por “e” ou “E” e um valor de expoente. Isso equivale a uma potência de 10 pela qual o número é multiplicado. Se o expoente for negativo (o que não é permitido para números inteiros, pois levariam a números fracionários), usa-se o sinal “-” antes do expoente. O sinal “+”, para expoentes positivos, é opcional. Assim, os números reais abaixo são equivalentes:

24.0e-03 24.0E-03 24.000e-3 0.024 0.024e+00

A.8 – ESTRUTURAS DE CONTROLE

Em VHDL-AMS há, como em linguagens de programação e descrição de hardware, estruturas utilizadas para tomar decisões baseadas em condições pré-estabelecidas. Notavelmente, as principais estruturas no contexto deste trabalho são:

- **`IF`** : executa as instruções contidas na estrutura apenas se determinada condição for satisfeita;
- **`CASE`**: lista, para certa condição, as alternativas possíveis, e define as instruções a serem executadas em cada caso;
- **`FOR`**: executa um conjunto de instruções, em seqüência, tantas vezes quanto for definido em sua declaração;
- **`WHILE`**: caso a condição definida seja satisfeita, executa uma ou mais instruções, em seqüência, repetidamente.

Tais estruturas são tipicamente seqüenciais As estruturas **`IF`** e **`CASE`**, entretanto, podem ser simultâneas. A diferença entre as estruturas é da forma descrita na tabela A.1. Note-se que, nas estruturas **`IF`**, os trechos **`ELSIF`** e **`ELSE`** são opcionais, e pode haver um número arbitrário de condições definidas por **`ELSIF`**. Nas estruturas **`CASE`**, deve haver um número finito de

possibilidades para o resultado da expressão utilizada, e todas devem ser abordadas na estrutura.

Tabela A.1 – estruturas IF e CASE seqüenciais e simultâneas

	Estrutura seqüencial	Estrutura simultânea

IF	**IF** *<condição>* **THEN**
	<declaração seqüencial>
	ELSIF *<condição>* **THEN**
	<declaração seqüencial>
	ELSE
	<declaração seqüencial>
	END IF;
		IF *<condição>* **USE**
		<declaração simultânea>
		ELSIF *<condição>* **USE**
		<declaração simultânea>
		ELSE
		<declaração simultânea>
		END USE;

CASE	**CASE** *<expressão>* **IS** **WHEN** *<escolhas>* **=>** *<declaração seqüencial>* **END CASE**;	**CASE** *<expressão>* **USE** **WHEN** *<escolhas>* **=>** *<declaração simultânea>* **END CASE**;

Quando estruturas **IF** ou **CASE** simultâneas forem utilizadas, é necessário informar ao simulador que há descontinuidades no comportamento analógico do bloco. Tais descontinuidades são modeladas no código através do comando **break**. Esse comando faz com que o cálculo dos valores analógicos do bloco seja reiniciado, tipicamente devido a alterações em outro objeto. Por exemplo, as linhas de código abaixo significam que a quantidade vi_i_out tem seu valor atrelado a vi_v_in somente se vi_on_off tiver o valor '1'; caso contrário, é nula. O comando **break** ao final faz com que, se vi_on_off mudar de valor, vi_i_out seja recalculada.

```
IF vi_on_off = '1' USE
      vi_i_out == vi_v_in-1.5*2.0e-4;
ELSE
      vi_i_out == 0.0;
END USE;
break on vi_on_off;
```

APÊNDICE B – RESTRIÇÕES DE NOMENCLATURA PARA HDL-AMS

Com o objetivo de facilitar a modelagem de blocos analógicos em linguagens de descrição de hardware diferentes, ou de utilizar blocos HDL-AMS em co-simulação entre linguagens diferentes, estão listadas a seguir restrições aos identificadores associados a objetos em VDHL-AMS e Verilog-AMS.

B.1 – VHDL-AMS

Um identificador em VHDL-AMS deve obedecer às seguintes regras:

- pode ser arbitrariamente longo;
- deve conter apenas letras do alfabeto (“A” a “Z” e “a” a “z”), dígitos decimais (“0” a “9”) e traço inferior (“_”);
- o primeiro caractere deve ser uma letra;
- o último caractere não pode ser o traço inferior;
- não pode conter dois traços inferiores consecutivos;
- letras maiúsculas e minúsculas são equivalentes;
- não podem ser iguais às palavras reservadas (listadas na Tabela B.1)

Identificadores estendidos podem conter qualquer seqüência de caracteres, não tendo que seguir as restrições acima. São escritos entre barras invertidas (“\”). Por exemplo, os identificadores estendidos abaixo são válidos em VHDL-AMS:

\next\ _sample&hold_\ \0__$! :)\

B.2 – VERILOG-AMS

Um identificador em Verilog-AMS deve obedecer às seguintes regras:

- pode ser arbitrariamente longo;
- pode conter letras, dígitos decimais, traço inferior e cifrão (“$”);
- o primeiro caractere deve ser uma letra ou traço inferior;
- letras maiúsculas e minúsculas são consideradas caracteres diferentes

Identificadores estendidos em Verilog-AMS começam com uma barra invertida (“\”) e terminam com espaço em branco, tabulação ou quebra de linha. Podem conter qualquer caractere.

B.3 – PALAVRAS RESERVADAS

A seguir estão listadas as palavras reservadas nas linguagens VHDL-AMS e Verilog-AMS. Para facilitar a leitura, a Tabela B.1 está organizada alfabeticamente, com espaços vazios em cada coluna onde for conveniente para manter a organização.

Tabela B.1 – palavras reservadas em Verilog-AMS e VHDL-AMS

Verilog-AMS	VHDL-AMS	Verilog-AMS	VHDL-AMS	Verilog-AMS	VHDL-AMS
above			body		disconnect
abs	abs	branch			downto
absdelay			break	driver_update	
ac_stim		buf			
	access		buffer	edge	
acos		bufif0		else	else
acosh		bufif1		end	end
	across		bus	endcase	
	after			endconnectrul es	
	alias	case	case	enddiscipline	
	all	casex		endfuction	
always		casez		endmodule	
analog		ceil		endnature	
analysis		cmos		endprimitive	
and	and		component	endspecify	
	architecture		configuration	endtable	
	array	connectrules		endtask	
asin			constant		entity
asinh		cos		event	
	assert	cosh		exclude	
assign		cross			exit
atan				exp	
atan2		ddt			
atanh		deassign			file
	attribute	default		final_step	
		defparam		flicker_noise	
begin	begin	disable		flow	
	block	discipline			for

Tabela B.1 – palavras reservadas em Verilog-AMS e VHDL-AMS (continuação)

Verilog-AMS	VHDL-AMS	Verilog-AMS	VHDL-AMS	Verilog-AMS	VHDL-AMS
force			label	nor	nor
forever		laplace_nd		not	not
fork		laplace_np		notif0	
from		laplace_zd		notif1	
function	function	laplace_zp			null
		large			
generate	generate	last_crossing			of
	generic		library		on
genvar		limexp			open
ground			limit	or	or
	group		linkage		others
	guarded		literal		out
		ln		output	
highz0		log			
highz1			loop		package
hypot				parameter	
		macromodule		pmos	
idt			map		port
idtmod		max		posedge	
if	if	medium			postponed
ifnone		min		potential	
	impure		mod	pow	
	in	module		primitive	
	inertial				procedural
inf		nand	nand		procedure
initial		nature	nature		process
initial_step		negedge			protected
inout	inout	net_resolution		pull0	
input			new	pull1	
integer			next	pulldown	
	is	nmos		pullup	
			noise		pure
join		noise_table			

Tabela B.1 – palavras reservadas em Verilog-AMS e VHDL-AMS (continuação)

Verilog-AMS	VHDL-AMS	Verilog-AMS	VHDL-AMS	Verilog-AMS	VHDL-AMS
	quantity		sll	triand	
		small		trior	
	range	specify		trireg	
rcmos		specparam			type
real			spectrum		
realtime		sqrt			unaffected
	range		sra		units
	reference		srl		until
reg		strong0			use
	register	strong1			
	reject		subnature		variable
release			subtype	vectored	
	rem	supply1			
repeat				wait	wait
	report	table		wand	
	return	tan		weak0	
rnmos		tanh		weak1	
	rol	task			when
	ror		terminal	while	while
rpmos			then	white_noise	
rtran			through	wire	
rtranif0		time			with
rtranif1		timer		wor	
			to	wreal	
scalared			tolerance		
	select	tran		xnor	xnor
	severity	tranif0		xor	xor
	shared	tranif1			
	signal	transition		zi_nd	
sin			transport	zi_np	
sinh		tri		zi_zd	
	sla	tri0		zi_zp	
slew		tri1			

APÊNDICE C – DOCUMENTAÇÃO DO CONVERSOR V-I

Neste trabalho, a documentação do conversor tensão-corrente será organizada como parte do Apêndice C. Pela metodologia proposta, seriam gerados documentos diferentes; estarão separados aqui seguindo o padrão de numeração do Apêndice. Para reduzir redundância neste trabalho, e como a intenção destes Apêndices é mais a de demonstrar a proposta, parte do conteúdo (especialmente figuras e tabelas) é omitida, sendo substituídapor uma indicação de onde o assunto já foi tratado neste trabalho. A numeração de páginas deste documento também será mantida. As folhas de rosto, que devem ser páginas separadas na documentação, aparecem aqui inseridas no texto.

C.1 – RESUMO

Documentação de Componente Virtual

RESUMO
(versão 1.0)

Fornecedor: Laboratório de Dispositivos e Circuitos Integrados (LDCI – UnB)
Produto: Conversor Tensão-Corrente

Versão: 1.1

Descrição:

O conversor tensão-corrente (VI) é um bloco de circuito de sinal-misto que converte linearmente um sinal analógico de tensão em um sinal analógico de corrente, preservando as informações contidas no sinal de entrada. É destinado a aplicações que necessitem processamento de sinais em modo de corrente.

Especificações principais:

Faixa de entrada: 1 V a 2 V

Faixa de saída: -100 µA a +100 µA

Faixa de freqüências: 0 a 25 kHz

Fornecido como: leiaute (*hard IP*)

Tecnologia de fabricação: CMOS 0,35 µm (C3B4C3 da *Austria Microsystems*)

Área: 0,022 mm^2 (217,33 µm x 100,28 µm)

C.2 – GUIA DO USUÁRIO

Folha de rosto

Documentação de Componente Virtual

GUIA DO USUÁRIO

(versão 1.0)

Fornecedor: Laboratório de Dispositivos e Circuitos Integrados (LDCI – UnB)
Produto: Conversor Tensão-Corrente
Versão: 1.1

Histórico de versões do documento

Versão	Data	Responsável
1.0	Julho de 2009	João Vitor B. Pimentel

Lista de Tabelas

C.2.1 – Introdução

Este documento trata da operação de um bloco de IP AMS. O bloco é um conversor tensão-corrente (V-I) linear com entrada analógica de tensão e três saídas analógicas de corrente. Inclui também entradas digitais para controle de operação. Neste documento são descritas especificações do bloco e suas funcionalidades.

As especificações do bloco são dadas na Tabela C.1.

Tabela C.1 – especificações de operação do bloco V-I

Especificações	
Tipo(s) de sinal	Tensão analógica
	Corrente analógica
	Sinais lógicos
Alimentação elétrica	3,3V (VDD) / 0,0 V (GND)
Faixa de entrada	1 V a 2 V
Faixa de saída	-100 µA a +100 µA
Freqüências de operação	0 a 25 kHz
Impedância de entrada	Alta (porta de transistor tipo N)
Impedância de saída	1 M Ω
Temperatura de operação	Entre 0 ºC e 70 ºC
Pinos de entrada	4 (1 analógico / 3 digitais)
Pinos de saída	3 (analógicos)
Modos de operação	4

A conversão do sinal de tensão ainda é realizada fora das faixas de entrada e de freqüências de operação, porém com degradação da linearidade.

C.2.2 – Estrutura do bloco

O circuito é composto por 4 sub-blocos, conforme ilustrado na Figura 4.4, chamados de *núcleo de conversão* (*núcleo*), *referência de corrente* (*referência*), *estágio intermediário de ganho* (*G4*) e *estágio de ganho e saída* (*G5*). Suas funções são as seguintes:

- Núcleo: converte o sinal de entrada em um sinal correspondente de corrente, na faixa de 2,5 µA a 12,5 µA;
- Referência: fornece corrente para a operação do *núcleo* (1,84 µA) e drena corrente (7,5 µA) do nó *n*, deslocando assim a faixa do sinal para -5 µA a +5 µA;
- G4: quadruplica a intensidade do sinal, mantendo a linearidade;
- G5: quintuplica a intensidade do sinal, mantendo a linearidade.

C.2.3 – Modos de operação

O V-I tem 4 modos de operação distintos. A escolha do modo de operação é feita pela combinação das entradas digitais, conforme descrito na Tabela 4.1. Os níveis baixo e alto das

entradas digitais são tensões aproximadamente iguais a 0 V e 3,3 V, respectivamente. Os modos de operação funcionam da seguinte maneira:

- Desligamento: o consumo do bloco não é cortado de fato. A entrada do *núcleo* é levada abaixo do seu limite inferior, independentemente da entrada, e a entrada do bloco G4 é desviada para o pino *vi_t_nr*, o que leva às seguintes leituras nos pinos de saída:
    - Saída pelo pino *vi_saida* : constante, de intensidade menor do que 0,2 μA;
    - Saída pelo pino *vi_t_nr* : constante, de intensidade 3,3 μA;
    - Saída pelo pino *vi_t_g4* : constante, de intensidade menor que que 0,2 μA.
- Teste NR : a saída de corrente no pino *vi_t_nr* é linearmente proporcional à entrada, porém, como os ganhos dos blocos G4 e G5 não foram aplicados, varia na faixa de ±5,0 μA. Este modo permite testar o conjunto *núcleo* + *referência* ao comparar-se a saída do pino *vi_t_nr* com a tensão aplicada na entrada do bloco.
- Teste G4 : a saída de corrente no pino *vi_t_g4* é linearmente proporcional à entrada, porém, como o ganho do bloco G5 não foi aplicado, varia na faixa de ±20,0 μA. Este modo, em conjunto com os resultados do Teste NR, permite testar o bloco *G4* ao comparar-se a saída do pino *vi_t_g4* com a tensão aplicada na entrada do bloco.
- Operação normal : a saída de corrente é dada pelo pino *vi_saida*, e se comporta conforme as especificações do bloco. Conhecidos os resultados dos modos Teste NR e Teste G4, este modo de operação permite verificar o funcionamento do bloco G5.

A mudança de estado do bloco causa sinais temporários (picos) nas saídas. Tais picos ficam dentro da faixa de saída do bloco. Se a transição entre estados levar pelo menos 20 μs (obedecendo à restrição de freqüência), os picos podem ser desprezados. Note-se também que os modos de teste mantêm as características do bloco, podendo ser utilizados normalmente caso suas faixas de saída sejam adequadas à faixa especificada pela aplicação.

C.2.4 – Problemas conhecidos

Em temperatura ambiente (27 ºC), a saída do bloco atinge efetivamente o valor máximo de 97,89 μA e o valor mínimo de 97,43 μA. Esse desvio da resposta ideal é considerado como um efeito indesejado da temperatura no sistema – que desloca a curva de resposta do bloco conforme mostrado na Figura 4.3 – e não como não-linearidade. O desvio da resposta dos modos de teste com a temperatura pode ser considerado equivalente ao desvio na saída, porém com a escala adequada à

saída do modo de teste.

C.3 – GUIA DE CRIAÇÃO

Folha de rosto

Documentação de Componente Virtual

GUIA DE CRIAÇÃO
(versão 1.0)

Fornecedor: Laboratório de Dispositivos e Circuitos Integrados (LDCI – UnB)
Produto: Conversor Tensão-Corrente
Versão: 1.1

Histórico de versões do documento

Versão	Data	Responsável
1.0	Julho de 2009	João Vitor B. Pimentel

Lista de Tabelas

C.3.1 – Introdução

Este documento trata da criação de um bloco de IP AMS. O bloco é um conversor tensão-corrente (V-I) linear com entrada analógica de tensão e três saídas analógicas de corrente. Inclui também entradas digitais para controle de operação. Neste documento são descritos os processos adotados durante o desenvolvimento do bloco até sua versão atual.

O conversor foi desenvolvido com o objetivo de minimizar tamanho e consumo. Sua topologia não utiliza amplificadores operacionais e difere de topologias tradicionais por fornecer saída de corrente simétrica sem a utilização de alimentação simétrica. Seu projeto foi realizado seguindo uma metodologia *top-down* com auxílio de plataformas de auxílio a projeto (CAD), utilizando modelo *BSIM3v3* para validação por simulação.

O bloco foi desenvolvido como parte de um sistema-em-chip, inicialmente (em sua versão 1.0) específico para dada aplicação – converter sinais de tensão de uma interface analógica para sinais de corrente adequados a um conversor analógico/digital operando em modo de corrente. O conversor A/D para o qual as especificações da versão 1.0 do V-I foram projetadas tem resolução de 8 bits na faixa de saída do V-I, resultando em um erro de quantização de 0,5 LSB de ±0,390625 μA.

Tabela C.2 – histórico de versões do bloco (V-I)

Versão	Data	Responsável	Alterações
1.0	Dezembro de 2008	Genival M. Araújo	-
1.1	Julho de 2009	João Vitor B. Pimentel	Inclusão de modos de teste

C.3.2 – Princípios de operação

O V-I pode ser compreendido como um conjunto de sub-blocos razoavelmente independentes – com a possível exceção do núcleo de conversão, que é alimentado pela referência, porém pode ser alimentado por fonte externa. A descrição dos bloco é dada no Guia do Usuário.

- Núcleo de conversão: o núcleo foi desenvolvido a partir de uma topologia que utiliza apenas transistores MOS. Uma referência de corrente é necessária para polarização dos circuitos. A saída de corrente é obtida através de um espelho de corrente.
- Referência de corrente: a referência de corrente é uma proposta de inovação sobre uma topologia anterior. Utiliza apenas um resistor para atingir boa estabilidade térmica. A referência contém o que podem ser considerados diferentes estágios de saída, cujo dimensionamento ajusta o valor da corrente a ser fornecida; assim, podem ser replicados ou ajustados para se obter outras saídas. Como foi projetada especificamente para o V-I, tem as duas saídas descritas no Guia do Usuário.
- Estágios de ganho: os blocos se baseiam em estruturas complementares de espelhos de corrente (espelhos P e espelhos N) para obter entrada e saída simétricas com

tensão de alimentação apenas positiva. O dimensionamento dos transistores ajusta os blocos para o ganho desejado. Para o V-I, foram utilizados dois estágios para que o ganho total fosse o desejado mantendo-se a linearidade (difícil de manter, para essa topologia, quando o ganho é muito grande).

C.4 – GUIA DE TESTE

Folha de rosto

Documentação de Componente Virtual

GUIA DE TESTE

(versão 1.0)

Fornecedor: Laboratório de Dispositivos e Circuitos Integrados (LDCI – UnB)
Produto: Conversor Tensão-Corrente
Versão: 1.1

Histórico de versões do documento

Versão	Data	Responsável
1.0	Julho de 2009	João Vitor B. Pimentel

Lista de Figuras

C.4.1 – Introdução

Este documento trata da validação de um bloco de IP AMS. O bloco é um conversor tensão-corrente (V-I) linear com entrada analógica de tensão e três saídas analógicas de corrente. Inclui também entradas digitais para controle de operação. Neste documento são descritos requisitos de teste e validação do bloco.

As entradas digitais controlam o modo em que o V-I opera, conforme a Tabela 5.2, que mostra tanto os modos de operação previstos para utilização quanto as demais combinações dos sinais de controle. Recomenda-se utilizar apenas os modos previstos.

C.4.2 – Estruturas de teste

A testabilidade do bloco foi implementada em sua versão 1.1, por meio das estruturas descritas na seção 4.1 deste trabalho. Consiste de chaves implementadas por transistores N e P, associados para obter condução ou corte complementarmente. Como os transistores são controlados pelo mesmo sinal de tensão, há, na transição, uma faixa de tensões em que não há complementaridade (um transistor conduzindo, outro em corte), mas ambos conduzem. Durante esta transição, pode haver sinais indesejados nas saídas. Porém o objetivo dos sinais de controle é selecionar o modo de operação, e não implementar um chaveamento rápido como em um circuito de amostragem.

Os sinais nas transições podem ser desconsiderados na aplicação do conversor (a não ser que gerem efeitos indesejados em outros blocos ao qual é interconectado) considerando-se a operação apenas com as tensões dos sinais de controle já em seu valor final. Se a transição dos sinais de controle obedecer aos requisitos de freqüência, o V-I opera como descrito.

C.4.3 – Condições para validação

A verificação do circuito por simulação deve ser feita utilizando-se modelos *BSIM3v3*, e considerando as características de processo da tecnologia citada no documento *Implementação Física* (seção C.5). Pinos de entrada e saída não devem ser deixados em aberto (sem sinal aplicado) ou com sinais fora das faixas especificadas no Guia do Usuário. A carga equivalente nas saídas prevista para a validação do bloco é uma combinação de resistências e capacitâncias, como mostrado na Figura C.1.

A temperatura considerada na validação deve estar dentro da faixa especificada, e ser mantida constante durante a validação. O bloco foi projetado para desempenho ótimo em 27 ºC.
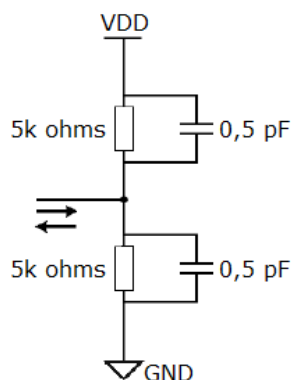
Figura C.1 – carga equivalente para validação do bloco

A alimentação elétrica deve ser capaz de suprir um consumo de 0,055 mW (rms). A validação realizada prevê variações de no máximo 10% na tensão de alimentação.

A validação do bloco levou em consideração a aplicação do bloco em uma interface analógica/digital, conforme descrito no Guia do Usuário. Por isso, considerou-se que erros ou distorções que provocassem variação menor do que 0,4 µA na(s) saída(s) não eram significativos.

C.5 – IMPLEMENTAÇÃO FÍSICA

Folha de rosto

Documentação de Componente Virtual

IMPLEMENTAÇÃO FÍSICA

(versão 1.0)

Fornecedor: Laboratório de Dispositivos e Circuitos Integrados (LDCI – UnB)

Produto: Conversor Tensão-Corrente

Versão: 1.1

Histórico de versões do documento

Versão	Data	Responsável
1.0	Julho de 2009	João Vitor B. Pimentel

Lista de Figuras

C.5.1 – Introdução

Este documento trata da implementação física (leiaute de máscaras) de um bloco de IP AMS. O bloco é um conversor tensão-corrente (V-I) linear com entrada analógica de tensão e três saídas analógicas de corrente. É fornecido como um leiaute otimizado e validado. Neste documento, é descrita a sua implementação como um conjunto de máscaras destinadas à fabricação.

C.5.2 – Descrição geométrica

O leiaute do circuito é mostrado na Figura 5.6. Uma representação geométrica é mostrada na Figura C.2, onde as proporções correspondem ao bloco implementado pelo leiaute, e os pinos indicam a posição destes no leiaute e os sinais aos quais devem ser conectados. De acordo com a descrição do bloco, os pinos *vi_vdd* e *vi_gnd* são de alimentação elétrica; os pinos *vi_vin*, *vi_t_nr*, *vi_t_g4* e *vi_saida* são de sinais analógicos; e *vi_d_cin*, *vi_d_cnr* e *vi_d_cg4* são de sinais digitais.

O circuito fabricado como descrito nesta documentação tem área de 0,022 mm^2 (217,33 μm x 100,28 μm).

C.5.3 – Tecnologia de fabricação

O bloco foi projetado, validado e prototipado utilizando a tecnologia *C35B4C3*, da *Austria Microsystems.* Os parâmetros utilizados são descritos em [54].
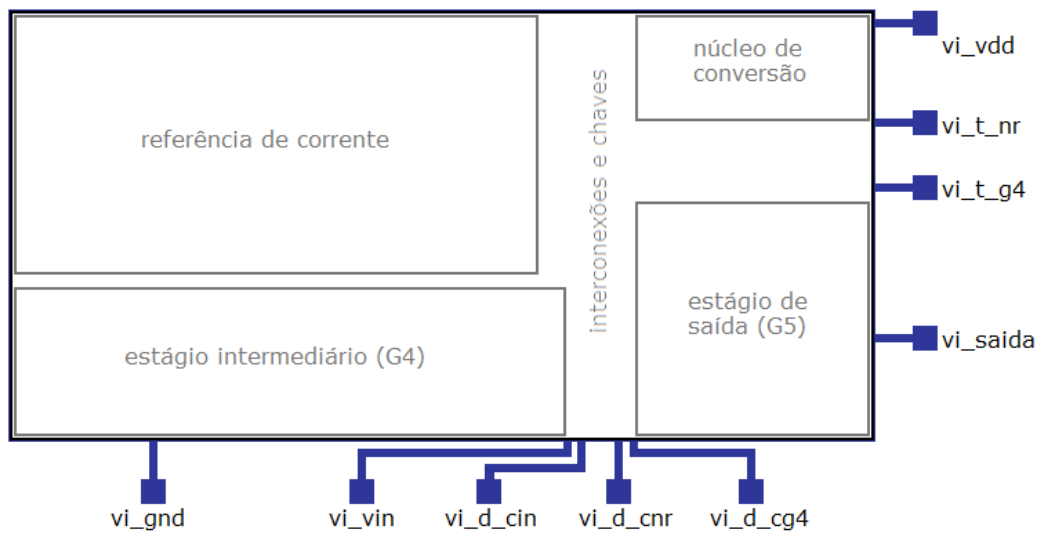
Figura C.2 – floorplan do conversor V-I

C.6 – MODELOS

Folha de rosto

Documentação de Componente Virtual

MODELOS

(versão 1.0)

Fornecedor: Laboratório de Dispositivos e Circuitos Integrados (LDCI – UnB)

Produto: Conversor Tensão-Corrente

Versão: 1.1

Histórico de versões do documento

Versão	Data	Responsável
1.0	Julho de 2009	João Vitor B. Pimentel

C.6.1 – Introdução

Este documento trata da modelagem de um bloco de IP AMS. O bloco é um conversor tensão-corrente (V-I) linear com entrada analógica de tensão e três saídas analógicas de corrente. É fornecido como um leiaute otimizado e validado. Neste documento, é descrita a modelagem de alto nível que permite a simulação e avaliação de seu funcionamento e desempenho.

Os modelos podem ser facilmente modificados com algum conhecimento da linguagem utilizada. Com as explicações deste documento e a documentação do IP, deve ser possível criar modelos equivalentes e avaliar corretamente as simulações.

Os códigos receberam a etiqueta de IP *soft* com os campos opcionais *Modelo* (funcional, comportamental ou estrutural) e *Sub-bloco* (no modelo estrutural, quando aplicável).

C.6.2 – Especificações dos modelos

Os modelos foram escritos na linguagem VHDL-AMS, de acordo com as especificações definidas em [29]. Os modelos foram descritos nos níveis descritos na Tabela 3.2, descrevendo características do comportamento do circuito de acordo com cada nível. Devido a algumas características específicas de cada nível, a interface do bloco pode ter sido mais ou menos detalhada, por isso as diferentes arquiteturas podem requerer diferentes entidades.

Todos os modelos utilizam os pacotes abertos e pré-definidos *ieee.math_real*, *ieee.electrical_systems* e *ieee.std_logic_1164* com suas configurações padrões.

As parametrizações utilizadas nos modelos envolvem portas genéricas declaradas na interface da entidade e também constantes definidas no corpo da arquitetura. Pode ser vantajoso definir uma constante como porta genérica, por exemplo para realizar-se simulações com várias instâncias da mesma arquitetura juntas, alterando algum valor de parâmetro. Isso pode ser feito facilmente manipulando-se o código fornecido, porém os códigos como estão satisfazem as simulações cujos resultados são apresentados, e não se garantem os resultados de qualquer modificação feita aos modelos.

C.6.3 – Condições de validação

Todos os códigos foram simulados com um conjunto de arquivos composto de:

- Código(s) VHDL-AMS no formato .vhms;
- Arquivo de entrada de comandos no formato .tcl;
- Arquivo de controle de simulação analógica no formato .scs;

As simulações foram realizadas utilizando as ferramentas NCVHDL, NCElab e NCSim – invocadas nesta ordem – da Cadence Design Systems. As simulações são executadas a partir da célula hierarquicamente superior – uma plataforma de teste específica para o modelo sendo simulada.

C.6.4 – Códigos

Os códigos para os modelos do V-I e suas plataformas de teste são aqueles apresentados no Apêndice E. Sua validação e resultados foram descritos nos capítulos 4 e 5.

APÊNDICE D – DOCUMENTAÇÃO DO CONVERSOR A/D

Assim como no Apêndice C, o que seriam documentos diferentes estão separados aqui seguindo o padrão de numeração do Apêndice. Também foram omitidos alguns dados já apresentados neste trabalho, uma vez que o objetivo destes apêndices é o de ilustrar a metodologia proposta, exemplificando o conteúdo da documentação de um IP.

D.1 – RESUMO

Documentação de Componente Virtual

RESUMO

(versão 1.0)

Fornecedor: Laboratório de Dispositivos e Circuitos Integrados (LDCI – UnB)
Produto: Conversor Analógico/Digital
Versão: 1.0

Descrição:

O conversor analógico/digital (A/D) é um bloco de circuito de sinal-misto que gera uma saída binária correspondente a um sinal analógico de corrente, convertendo as informações contidas no sinal a um formato digital. É destinado a aplicações que necessitem processamento digital de sinais analógicos.

Especificações principais:

Faixa de entrada: -100 µA a +100 µA
Saída: 256 palavras de 8 bits, entre 0000 0000 e 1111 1111
Freqüência de amostragem: 50 mil amostras por segundo (50 ksps)
Fornecido como: leiaute (*hard IP*)
Tecnologia de fabricação: CMOS 0,35 µm (C3B4C3 da *Austria Microsystems*)
Área: 0,61 mm^2

D.2 – GUIA DO USUÁRIO

Folha de rosto

Documentação de Componente Virtual

GUIA DO USUÁRIO

(versão 1.0)

Fornecedor: Laboratório de Dispositivos e Circuitos Integrados (LDCI – UnB)

Produto: Conversor Analógico/Digital

Versão: 1.0

Histórico de versões do documento

Versão	Data	Responsável
1.0	Julho de 2009	João Vitor B. Pimentel

Lista de Tabelas

D.2.1 – Introdução

Este documento trata da operação de um bloco de IP AMS. O bloco é um conversor analógico/digital (A/D) cíclico com entrada analógica de corrente e saída digital. Sua operação é controlada por um conjunto de sinais digitais. Neste documento são descritas especificações do bloco e suas funcionalidades.

As especificações do bloco são dadas na Tabela D.1. Alguns dos pinos considerados na Tabela são opcionais. Seu uso será explicado nas seções seguintes.

Tabela D.1 – especificações de operação do bloco

Especificações	
Tipo(s) de sinal	Corrente analógica
	Sinais lógicos
Alimentação elétrica	5,0 V / 3,3 V / 0,0 V digital
	3,3 V / 0,0 V analógica
Faixa de entrada	-100 µA a +100 µA / 0,8 V a 2,1 V
Saída	Palavra digital de 8 bits
Erro de quantização	±0,390625 µA (0,5 LSB)
Faixas de freqüências	0 a 25 kHz para o sinal de entrada
	16 MHz para o sinal de relógio digital
Freqüência de amostragem	50 ksps
Impedância de entrada	2,4 kΩ
Temperatura de operação	Entre 0 ºC e 70 ºC
Pinos de entrada	9 (4 analógicos / 5 digitais)
Pinos de saída	5 (3 analógicos / 2 digitais)
Modos de operação	2

As entradas e saídas digitais descritas nesta documentação refere-se à interface digital que pode ser controlada externamente. Há um sub-bloco de circuitaria digital (controle) associado ao A/D, cuja operação é descrita em documentação separada.

D.2.2 – Estrutura do bloco

O VC é composto de um bloco principal (chamado de *ad_nucleo*), responsável pela conversão do sinal analógico, e dois blocos auxiliares de uso opcional, *ad_vi* e *ad_memo_ext*, que têm por função prover sinais necessários ao A/D, mas que podem ser gerados externamente. Na Tabela D.2 são listados os pinos do A/D, com breve descrição. Foram omitidos da Tabela pinos exclusivos do controle digital.

O núcleo pode ser entendido como uma estrutura composta por 5 blocos, como mostrado na Figura 4.6, funcionando em ciclos. O sub-bloco *S/H* é responsável pela amostragem da entrada de corrente; o sub-bloco *Comp* é um comparador com saída binária; os sub-blocos *Memo* e *Ref* fornecem a entrada do comparador, dependendo do resultado do ciclo anterior. O funcionamento do núcleo será detalhado na próxima seção.

O bloco *ad_vi* converte sinais de tensão em uma faixa de 100mV em sinais de corrente na faixa de entrada do A/D. Embora possa ser utilizado na operação do A/D, este bloco é destinado principalmente à caracterização do VC, sendo recomendado o uso de sinais de corrente externos (ou provenientes de outro circuito) como entrada do núcleo do A/D.

O bloco *ad_memo_ext* tem funcionamento similar a *ad_vi*. Seu propósito é fornecer a corrente de referência necessária ao funcionamento do A/D, e seu uso é opcional, podendo ser substituído por uma fonte externa ou proveniente de outro circuito.

Tabela D.2 – Pinos do VC

Pino	Entrada (E)/Saída (S)	Descrição
ad_a_input	E	Entrada analógica de corrente
ad_a_iref	E	Entrada de referência de corrente
ad_a_smp	S	Saída da corrente amostrada
ad_a_vi_vin1	E	Entrada do bloco *ad_vi*
ad_a_vi_iout1	S	Saída do bloco *ad_vi*
ad_a_vi_vin2	E	Entrada do bloco *ad_memo_ext*
ad_a_vi_iout2	S	Saída do bloco *ad_memo_ext*
ad_d_on_off	E	Sinal de liga/desliga
ad_d_start	E	Sinal de início de conversão
ad_d_reset	E	Sinal de reinicialização
ad_d_saida	S	Resultado digital da conversão
ad_d_eoc	S	Indicação de fim de conversão
ad_d_clk	E	Sinal digital de relógio
ad_d_cc	E	Sinal de conversão contínua
ad_a_vdda	E/S	Alimentação analógica (3,3 V)
ad_a_gnda	E/S	Alimentação analógica (0 V)
ad_vdd5v	E/S	Alimentação da seção digital (5 V)
ad_vdd3v	E/S	Alimentação da seção digital (3,3 V)
ad_gndd	E/S	Alimentação da seção digital (0 V)

D.2.3 – Modos de operação

O A/D pode operar em dois modos distintos: o modo de operação normal e o modo de teste. O modo de teste é acionado pelo pino digital *MEM_SEL* e tem suas entradas e saídas próprias, todas ativadas e lidas pelo controle digital – descrito em documentação própria.

No modo de operação normal, a corrente de entrada amostrada em um instante t tem seu valor convertido em uma palavra digital de 8 bits, proporcional ao valor da corrente em relação à faixa de entrada. O A/D entrega, serialmente, 8 bits na saída correspondentes ao resultado da conversão. A saída começa pelo MSB e cada bit posterior é entregue em um flanco de subida (i.e., transição de nível baixo para nível alto) do relógio digital externo. Um ciclo de conversão completo, da amostragem até o fim da palavra serial, dura 20 μs.

D.2.3.1 – Detalhamento do modo de operação normal

A operação do A/D se dá com o pino *ad_d_on_off* em nível alto ('1') e uma corrente constante de 100 μA no pino *ad_a_iref*. A inicialização é feita levando-se o pino *ad_d_reset* a nível baixo ('0') e, em seguida, o pino *ad_d_start* a nível alto. Na subida de relógio seguinte, é iniciado o ciclo de conversão, com a amostragem da corrente presente na entrada *ad_a_input*. A partir de 1,5 μs a amostragem, a corrente amostrada fica disponível no pino *ad_a_smp* até o fim do ciclo de conversão. 18 μs após a corrente ser apresentada no pino *ad_a_smp*, 8 bits são entregues na saída digital *ad_d_saida*, um a cada flanco de subida do sinal *ad_d_clk* – totalizando a duração de 20 μs para o ciclo de conversão.

Ao fim da palavra de saída, o sinal *ad_d_eoc* vai a '1'. Caso o sinal *ad_d_cc* esteja em nível alto, imediatamente após o fim do ciclo de conversão, a corrente é amostrada novamente e o processo se repete.

D.2.4 – Problemas conhecidos

Ainda não foi feita uma caracterização suficiente do A/D para detalhar os efeitos não-ideais de seu funcionamento. Porém, alguns resultados já devem ser observados.

Testes preliminares com o A/D mostraram que a faixa de entrada real pode ser mais restrita

do que a faixa ideal. Os extremos da entrada são considerados pelo A/D em funcionamento como algo em torno de 98 μA (variando levemente com mudanças de temperatura); isso significa que o maior valor digital (1111 1111) será atribuído a um valor de corrente mais baixo do que o máximo da entrada. Isso equivale a um erro de ganho, conforme descrito no item 4.2.1.2.

Outro problema detectado foi quanto ao uso do sinal de conversão contínua, *ad_d_cc*, que causa erros na conversão. A operação, no entanto, funciona adequadamente se, ao fim de uma conversão, o sinal *ad_d_reset* for levado a nível alto e sem seguida novamente a nível baixo.

D.3 – GUIA DE CRIAÇÃO

Folha de rosto

Documentação de Componente Virtual

GUIA DE CRIAÇÃO

(versão 1.0)

Fornecedor: Laboratório de Dispositivos e Circuitos Integrados (LDCI – UnB)
Produto: Conversor Analógico/Digital
Versão: 1.0

Histórico de versões do documento

Versão	Data	Responsável
1.0	Julho de 2009	João Vitor B. Pimentel

Lista de Figuras

D.3.1 – Introdução

Este documento trata da criação de um bloco de IP AMS. O bloco é um conversor analógico/ digital (A/D) cíclico com entrada analógica de corrente e saída digital. Sua operação é controlada por um conjunto de sinais digitais. Neste documento são descritos os processos adotados durante o desenvolvimento do bloco até sua versão atual.

O conversor foi desenvolvido com o objetivo de minimizar tamanho e consumo. Sua topologia não é superior a outras topologias conhecidas em termos de velocidade de conversão, porém tem consumo baixo e utiliza pouca área em um chip, sendo útil para aplicações de baixo custo que não tenham requisitos altos de velocidade. Seu projeto foi realizado seguindo uma metodologia *top-down* com auxílio de plataformas de auxílio a projeto (CAD), utilizando modelo *BSIM3v3* para validação por simulação.

A versão atual do bloco é a versão 1.0, que foi prototipada e está sendo caracterizada.

D.3.2 – Princípios de operação

O A/D foi desenvolvido baseando-se na utilização de células copiadoras de corrente. Células desse tipo, controladas por chaves, utilizam transistores e um capacitor para amostrar uma corrente e manter seu valor durante o tempo necessário. A operação em modo de corrente chaveada em uma topologia cíclica permitiu a diminuição da complexidade e do tamanho físico do circuito, com células básicas realizando operações repetidamente sobre um sinal de corrente para chegar ao resultado final.
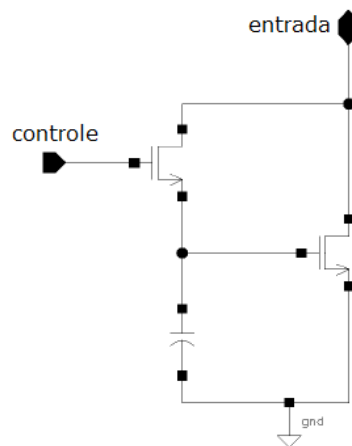
Figura D.1 – Célula copiadora de corrente

No projeto do bloco, foram utilizadas técnicas de projeto (elétrico e de leiaute) e estruturas para minimizar efeitos indesejados especialmente em circuitos chaveados, como a injeção de cargas resultante do rápido chaveamento.

D.4 – GUIA DE TESTE

Folha de rosto

Documentação de Componente Virtual

GUIA DE TESTE

(versão 1.0)

Fornecedor: Laboratório de Dispositivos e Circuitos Integrados (LDCI – UnB)
Produto: Conversor Analógico/Digital
Versão: 1.0

Histórico de versões do documento

Versão	Data	Responsável
1.0	Julho de 2009	João Vitor B. Pimentel

D.4.1 – Introdução

Este documento trata da validação de um bloco de IP AMS. O bloco é um conversor analógico/digital (A/D) cíclico com entrada analógica de corrente e saída digital. Sua operação é controlada por um conjunto de sinais digitais. Neste documento são descritos requisitos de teste e validação do bloco.

D.4.2 – Estruturas de teste

Há um modo de teste para o VC, implementado através do controle digital, descrito em documentação separada. Porém, para caracterizar a funcionalidade do A/D, basta verificar o seu funcionamento,d e acordo com o descrito no Guia do Usuário, com os três blocos (*ad_nucleo*, *ad_vi* e *ad_memo_ext*) operando juntos. O uso das duas últimas na caracterização do núcleo ficará evidente pela descrição de seu funcionamento, a seguir.

D.4.2.1 – Bloco *ad_vi*

Para caracterizar o bloco, deve-se alimentá-lo adequadamente (conectando-o aos pinos *ad_a_vdda* e *ad_a_gnda*) e curto-circuitar sua entrada e saída (pinos *ad_a_vi_vin1* e *ad_a_vi_iout1*). Mede-se a tensão (chamada de *vi_med*) que será utilizada na caracterização. Para encontrar as tensões de entrada que levam aos extremos de saída (±100 μA), conectam-se duas resistências em torno de 5 kΩ cada, em série entre VDD e GND, de modo que a tensão no nó entre as resistências seja igual a *vi_med*. Em seguida, liga-se ao nó a saída *ad_a_vi_iout1* para obter-se os valores máximo, mínimo e central da resposta do bloco. Variando uma tensão entre os extremos, pode-se caracterizar a linearidade de *ad_vi*. O uso principal de *ad_vi*, uma vez caracterizado, é de fornecer correntes ao núcleo do A/D, varrendo toda a faixa de entrada para a observação da saída.

D.4.2.2 – Bloco *ad_memo_ext*

A função do bloco *ad_memo_ext* é a de fornecer uma corrente de referência ao A/D. O bloco funciona basicamente como o *ad_vi*, sendo sua caracterização análoga. Porém, para caraterizá-lo suficientemente para seu propósito basta medir a tensão de entrada para a qual a saída é de 100 μA.

D.4.3 – Condições para validação

A verificação do circuito por simulação deve ser feita utilizando-se modelos *BSIM3v3*, e considerando as características de processo da tecnologia citada no documento *Implementação Física* (seção D.5). Pinos de entrada e saída não devem ser deixados em aberto (sem sinal aplicado), ou com sinais fora das faixas especificadas no Guia do Usuário (exceto aqueles dos blocos *ad_vi* e *ad_memo_ext*, que não precisam ser utilizados e podem ser mantidos aterrados).

A temperatura considerada na validação deve estar dentro da faixa especificada, e ser mantida constante durante a validação. O bloco foi projetado para desempenho ótimo em 27 ºC. A alimentação elétrica deve ser capaz de suprir um consumo de 2 mW.

Conforme descrito no Guia do Usuário, recomenda-se deixar o sinal *ad_cc* em nível baixo durante toda a caracterização, inciando novos ciclos por meio dos pinos *ad_d_start* e *ad_d_reset*.

D.5 – IMPLEMENTAÇÃO FÍSICA

Folha de rosto

Documentação de Componente Virtual

IMPLEMENTAÇÃO FÍSICA
(versão 1.0)

Fornecedor: Laboratório de Dispositivos e Circuitos Integrados (LDCI – UnB)
Produto: Conversor Analógico/Digital
Versão: 1.0

Histórico de versões do documento

Versão	Data	Responsável
1.0	Julho de 2009	João Vitor B. Pimentel

Lista de Figuras

D.5.1 – Introdução

Este documento trata da implementação física de um bloco de IP AMS. O bloco é um conversor analógico/digital (A/D) cíclico com entrada analógica de corrente e saída digital. Sua operação é controlada por um conjunto de sinais digitais. Neste documento, é descrita a sua implementação como um conjunto de máscaras destinadas à fabricação. O leiaute dos blocos do circuito é mostrado na Figura D.2
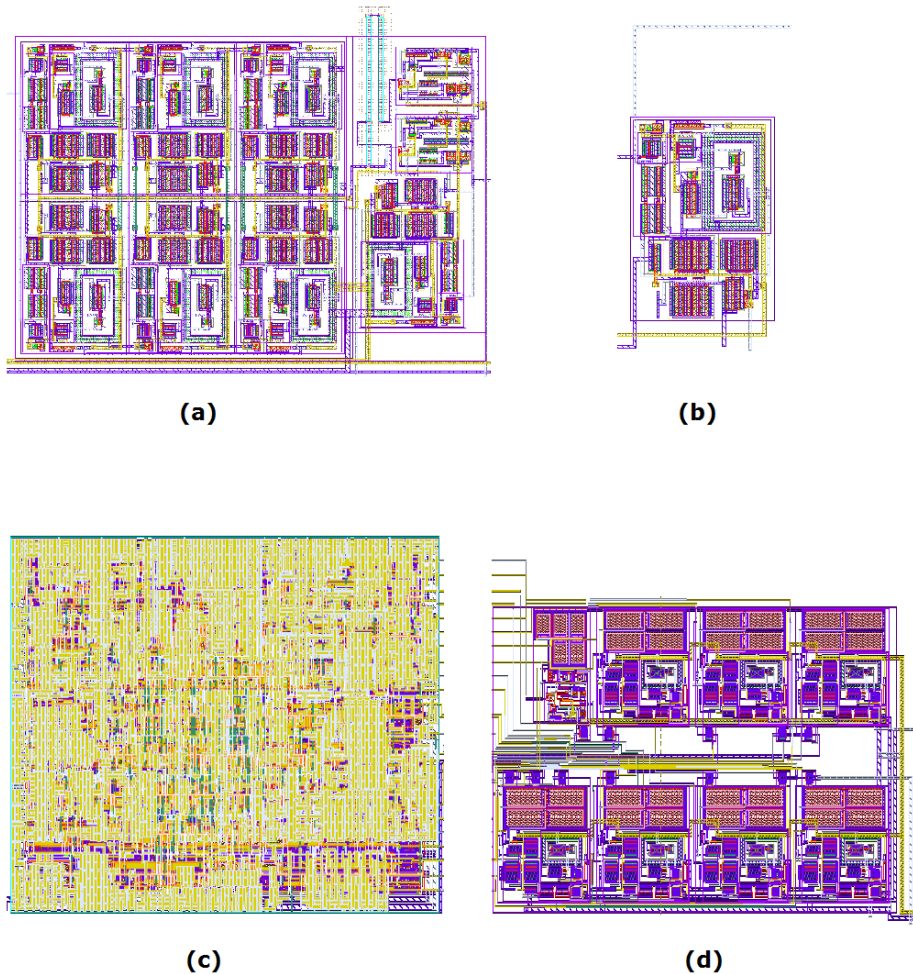
Figura D.2 – leiaute do circuito (a) *ad_memo_ext*; (b) *ad_vi*; (c) controle digital; (e) núcleo

D.5.2 – Descrição geométrica

Uma representação geométrica da implementação física do A/D é mostrada na Figura D.3, onde as proporções correspondem ao bloco implementado pelo leiaute. Para informações sobre os demais pinos do controle digital (parte superior esquerda da Figura D.3), referir-se à documentação da seção digital.
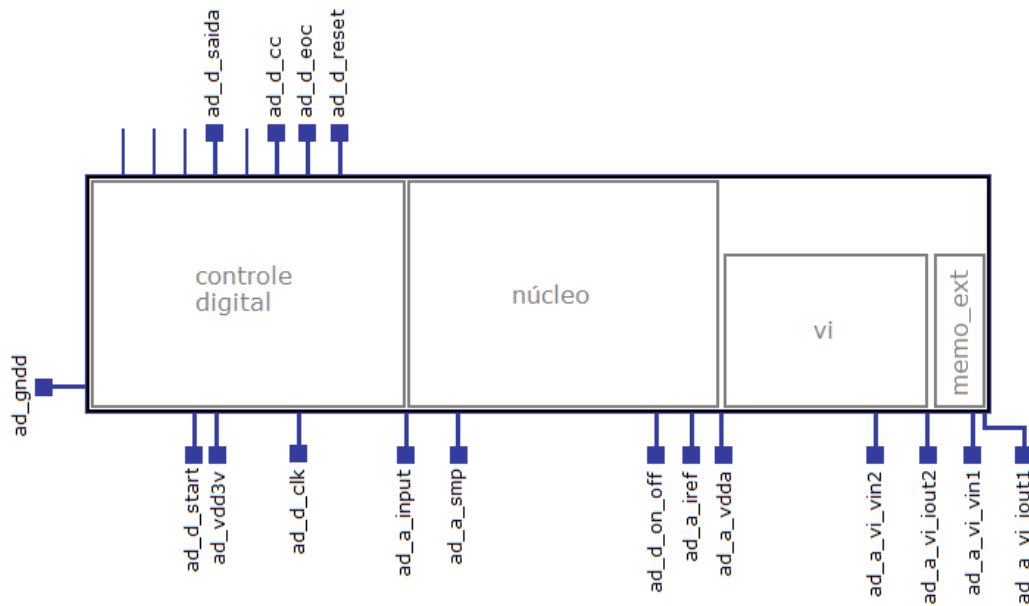
Figura D.3 – *floorplan* do conversor A/D

D.5.3 – Tecnologia de fabricação

O bloco foi projetado, validado e prototipado utilizando a tecnologia *C35B4C3*, da *Austria Microsystems.* Os parâmetros utilizados são descritos em [54].
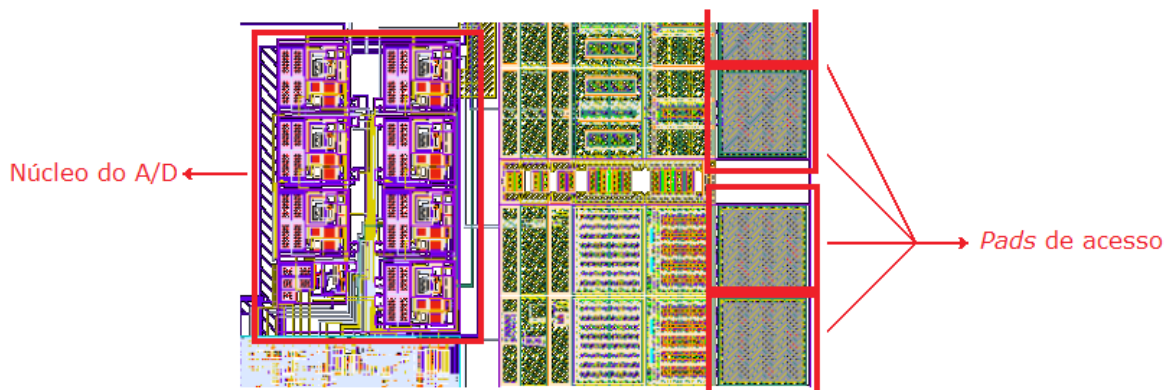
Figura D.4 – proporção entre circuito e *pads* de acesso

Na Figura D.3, os pinos indicam a posição destes no leiaute e os sinais aos quais devem ser conectados; porém, devido às dimensões do bloco na tecnologia utilizada para fabricação, o uso de *pads* (conexões externas no chip) em um chip fabricado deve superar razoavelmente a descrição mostrada aqui, que indica apenas as conexões que devem ser feitas aos pads. A proporção entre o circuito e pads utilizados no protótipo pode ser vista na Figura D.4.

D.6 – MODELOS

Folha de rosto

Documentação de Componente Virtual

MODELOS

(versão 1.0)

Fornecedor: Laboratório de Dispositivos e Circuitos Integrados (LDCI – UnB)
Produto: Conversor Analógico/Digital
Versão: 1.0

Histórico de versões do documento

Versão	Data	Responsável
1.0	Julho de 2009	João Vitor B. Pimentel

D.6.1 – Introdução

Este documento trata da modelagem de um bloco de IP AMS. O bloco é um conversor analógico/digital (A/D) cíclico com entrada analógica de corrente e saída digital. Sua operação é controlada por um conjunto de sinais digitais. Neste documento, é descrita a modelagem de alto nível que permite a simulação e avaliação de seu funcionamento e desempenho.

Os modelos podem ser facilmente modificados com algum conhecimento da linguagem utilizada. Com as explicações deste documento e a documentação do IP, deve ser possível criar modelos equivalentes e avaliar corretamente as simulações.

Os códigos receberam a etiqueta de IP *soft* com os campos opcionais *Modelo* (funcional, comportamental ou estrutural) e *Sub-bloco* (no modelo estrutural, quando aplicável).

D.6.2 – Especificações dos modelos

Os modelos foram escritos na linguagem VHDL-AMS, de acordo com as especificações definidas em [29]. Os modelos foram descritos nos níveis descritos na Tabela 3.2, descrevendo características do comportamento do circuito de acordo com cada nível. Devido a algumas características específicas de cada nível, a interface do bloco pode ter sido mais ou menos detalhada, por isso as diferentes arquiteturas podem requerer diferentes entidades.

Todos os modelos utilizam os pacotes abertos e pré-definidos *ieee.math_real*, *ieee.electrical_systems* e *ieee.std_logic_1164* com suas configurações padrões.

As parametrizações utilizadas nos modelos envolvem portas genéricas declaradas na interface da entidade e também constantes definidas no corpo da arquitetura. Pode ser vantajoso definir uma constante como porta genérica, porém os códigos como estão satisfazem as simulações cujos resultados são apresentados, e não se garantem os resultados de qualquer modificação feita aos modelos.

D.6.3 – Condições de validação

Todos os códigos foram simulados com um conjunto de arquivos composto de:

- Código(s) VHDL-AMS no formato .vhms;
- Arquivo de entrada de comandos no formato .tcl;
- Arquivo de controle de simulação analógica no formato .scs;

As simulações foram realizadas utilizando as ferramentas NCVHDL, NCElab e NCSim – invocadas nesta ordem – da Cadence Design Systems. As simulações são executadas a partir da célula hierarquicamente superior – uma plataforma de teste específica para o modelo sendo simulada.

D.6.4 – Códigos

Os códigos para os modelos do V-I e suas plataformas de teste são aqueles apresentados no Apêndice E. Sua validação e resultados foram descritos nos capítulos 4 e 5.

APÊNDICE E – CÓDIGOS VHDL-AMS

Os códigos VHDL-AMS usados nos modelos serão apresentados neste Apêndice. O uso de cores diferentes visa facilitar a compreensão dos códigos ao destacar palavras reservadas, tipos de objetos e destacar o conteúdo de comentários, que não é lido pelo compilador. Onde tiver sido necessário, neste documento, incluir quebras de linha que não fazem parte do código, elas estarão indicadas pelo símbolo "»" no começo da nova linha.

As plataformas de teste não têm interface externa, ou seja, a declaração de entidade não lista portas. Por isso, suas declarações de entidade (à exceção do nome) são iguais à que será apresentada no item E.1.1.1; para as demais plataformas de teste, serão apresentadas apenas as declarações de arquitetura. Também foram omitidos, após o item E.1.1, as declarações uso de biblioteca – que são iguais para todas as entidades – e os comandos `limit`.

E.1 – MODELAGEM DO CONVERSOR V-I

E.1.1 – Modelo funcional

```
LIBRARY ieee;
USE ieee.electrical_systems.ALL;
USE ieee.math_real.ALL;
USE ieee.std_logic_1164.ALL;

LIBRARY worklib;
USE worklib.ALL;

entity vi_modelo is
      generic (temp_c : real := 27.0);
      port (terminal vi_vin : electrical;
            terminal vi_saida : electrical;
            terminal vi_t_nr : electrical;
            terminal vi_t_g4 : electrical;
            signal vi_d_cin : in bit;
            signal vi_d_cnr : in bit;
            signal vi_d_cg4 : in bit);
end entity vi_modelo;

-----------------------------------------------------------------------
-- Os modelos funcional e comportamental têm as mesmas portas. Diferem na
-- descrição da arquitetura. Por isso a temperatura é incluída na decla-
-- ração da entidade, mesmo não sendo utilizada no modelo funcional.
-- A temperatura é incluída na declaração da entidade, mesmo não sendo
```

```
-- utilizada no modelo funcional, para que a interface da entidade seja
-- válida tanto para o modelo funcional quanto para o comportamental.
-----------------------------------------------------------------------
-- Etiqueta_IP_soft % Fornecedor LDCI-UnB % Produto conversor_vi % Versao 1.1
»     % Modelo funcional
-------------------------------------------------------------------------
architecture vi_funcional of vi_modelo is

      -- Declaração de tensões e correntes nos terminais:
      quantity vi_v_in across vi_i_in through vi_vin;
      quantity vi_i_saida through vi_saida;
      quantity vi_i_tnr through vi_t_nr;
      quantity vi_i_tg4 through vi_t_g4;

      -- Comandos utilizados para definir o passo máximo utilizado pelo
      -- simulador no cálculo do comportamento analógico do bloco:
      limit all : voltage with 0.000001;
      limit all : current with 0.000001;


begin

      vi_v_in == vi_i_in*1.0e+6;

      IF vi_d_cin = '1' USE

            IF vi_d_cnr = '1' and vi_d_cg4 = '1' USE
            -- Modo de operação normal --
                  vi_i_saida == (vi_v_in-1.5)*2.0e-4;
                  vi_i_tnr == 0.0;
                  vi_i_tg4 == 0.0;

            ELSIF vi_d_cnr = '0' USE
            -- Modo de teste NR --
                  vi_i_saida == 0.0;
                  vi_i_tnr == (vi_v_in-1.5)*1.0e-5;
                  vi_i_tg4 == 0.0;

            ELSE
            -- Modo de teste G4 --
                  vi_i_saida == 0.0;
                  vi_i_tnr == 0.0;
                  vi_i_tg4 == (vi_v_in-1.5)*4.0e-5;
            END USE;

      ELSE
      -- Desligamento --
            vi_i_saida == 0.0;
            vi_i_tnr == 0.0;
            vi_i_tg4 == 0.0;
      END USE;

      break on vi_d_cin, vi_d_cnr, vi_d_cg4;

end architecture vi_funcional;

-----------------------------------------------------------------------
```

E.1.1.1 – Testbench

```
entity vi_funcional_tb is
end entity vi_funcional_tb;


architecture vi_funcional_tb of vi_funcional_tb is

	constant freq_v : real :=22.0e3;    -- freqüência do sinal de entrada
	                                    -- (no teste, 22 kHz)

	signal TB_D_CIN, TB_D_CNR, TB_D_CG4 : bit;

	terminal TB_A_INPUT : electrical;
		quantity tb_v_in across tb_i_in through TB_A_INPUT;

	terminal TB_A_TNR, TB_A_TG4, TB_A_SAIDA : electrical;


begin

	-- Mapeia a entidade sob teste:
	vi : entity work.vi_modelo(vi_funcional)
		port map (vi_vin => TB_A_INPUT,
		        vi_saida => TB_A_SAIDA,
		        vi_t_nr => TB_A_TNR,
		        vi_t_g4 => TB_A_TG4,
		        vi_d_cin => TB_D_CIN,
		        vi_d_cnr => TB_D_CNR,
		        vi_d_cg4 => TB_D_CG4);

	-- Gera estímulo (tensão senoidal):
	tb_v_in == 1.5+0.5*sin(2.0*math_pi*freq_v*now);

	break on TB_D_CIN, TB_D_CNR, TB_D_CG4;

	operacao : process is
	begin

		-- Inicia modo de operação normal --
		TB_D_CIN <= '1';
		TB_D_CNR <= '1';
		TB_D_CG4 <= '1';
			wait for 20 us;

		-- Inicia modo de teste NR --
		TB_D_CIN <= '1';
		TB_D_CNR <= '0';
		TB_D_CG4 <= '1';
			wait for 20 us;

		-- Inicia modo de teste G4 --
		TB_D_CIN <= '1';
		TB_D_CNR <= '1';
		TB_D_CG4 <= '0';
			wait for 20 us;

		-- Modo de desligamento --
		TB_D_CIN <= '0';
		TB_D_CNR <= '0';
```

```
        TB_D_CG4 <= '0';
              wait for 20 us;

        -- Retorna ao modo de operação normal --
        TB_D_CIN <= '1';
        TB_D_CNR <= '1';
        TB_D_CG4 <= '1';

        wait;
    end process operacao;

end architecture vi_funcional_tb;
---------------------------------------------------------------------------
```

E.1.2 – Modelo comportamental

```
---------------------------------------------------------------------------
-- Etiqueta_IP_soft % Fornecedor LDCI-UnB % Produto conversor_vi % Versao 1.1
»      % Modelo comportamental
---------------------------------------------------------------------------
architecture vi_comportamental of vi_modelo is
      quantity vi_v_in across vi_i_in through vi_vin;
      quantity vi_v_saida across vi_i_saida through vi_saida;

      quantity vi_v_tnr across vi_i_tnr through vi_t_nr;

      quantity vi_v_tg4 across vi_i_tg4 through vi_t_g4;

      constant coef_t00_27 : real := (195.32 + (27.0 - temp_c)/1.697)*1.0e-06;
      constant coef_t27_70 : real := (195.32 + (27.0 - temp_c)/1.915)*1.0e-06;

begin

      vi_v_in == vi_i_in*1.0e+6;

      cin : IF vi_d_cin = '1' USE

            modo : IF vi_d_cnr = '1' and vi_d_cg4 = '1' USE
            -- Modo de operação normal  --
                  temp_111 : IF temp_c > 27.0 USE
                        vi_i_saida == (vi_v_in - 0.95189)*coef_t27_70
»                              - 106.827e-06;
                        vi_i_tnr == 0.0;
                        vi_i_tg4 == 0.0;
                  ELSE
                        vi_i_saida == (vi_v_in - 0.86801)*coef_t00_27
»                              - 123.210e-06;
                        vi_i_tnr == 0.0;
                        vi_i_tg4 == 0.0;
                  END USE temp_111;

            ELSIF vi_d_cnr = '0' USE
            -- Modo de teste NR --
                  temp_101 : IF temp_c > 27.0 USE
                        vi_i_saida == 0.0;
                        vi_i_tnr == ((vi_v_in - 0.95189)*coef_t27_70
»                              - 106.827e-06)/20.0;
                        vi_i_tg4 == 0.0;
```

```
                  ELSE
                        vi_i_saida == 0.0;
                        vi_i_tnr == ((vi_v_in - 0.86801)*coef_t00_27
»                              - 123.210e-06)/20.0;
                        vi_i_tg4 == 0.0;
                  END USE temp_101;

            ELSE
            -- Modo de teste G4 --
                  temp_110 : IF temp_c > 27.0 USE
                        vi_i_saida == 0.0;
                        vi_i_tnr == 0.0;
                        vi_i_tg4 == ((vi_v_in - 0.95189)*coef_t27_70
»                              - 106.827e-06)/5.0;
                  ELSE
                        vi_i_saida == 0.0;
                        vi_i_tnr == 0.0;
                        vi_i_tg4 == ((vi_v_in - 0.86801)*coef_t00_27
»                              - 123.210e-06)/5.0;
                  END USE temp_110;

            END USE;

      ELSE
      -- Desligamento --
            vi_i_saida == 0.0;
            vi_i_tnr == 0.0;
            vi_i_tg4 == 0.0;
      END USE cin;

      break on vi_d_cin, vi_d_cnr, vi_d_cg4;

end architecture vi_comportamental;
--------------------------------------------------------------------------
```

E.1.2.1 – Testbench

```
architecture vi_comportamental_tb of vi_comportamental_tb is
      constant freq_v : real :=22.0e3;    -- freqüência do sinal de entrada

      signal TB_D_CIN, TB_D_CNR, TB_D_CG4 : bit;

      terminal TB_A_INPUT : electrical;
            quantity tb_v_in across tb_i_in through TB_A_INPUT;

      terminal TB_A_TNR, TB_A_TG4 : electrical;

      -- São criados 9 terminais de saída, um para cada temperatura:
      terminal TB_A_SAIDA_27 : electrical;     constant t27 : real := 27.0;
      terminal TB_A_SAIDA_00 : electrical;     constant t00 : real := 0.0;
      terminal TB_A_SAIDA_10 : electrical;     constant t10 : real := 10.0;
      terminal TB_A_SAIDA_20 : electrical;     constant t20 : real := 20.0;
      terminal TB_A_SAIDA_30 : electrical;     constant t30 : real := 30.0;
      terminal TB_A_SAIDA_40 : electrical;     constant t40 : real := 40.0;
      terminal TB_A_SAIDA_50 : electrical;     constant t50 : real := 50.0;
      terminal TB_A_SAIDA_60 : electrical;     constant t60 : real := 60.0;
      terminal TB_A_SAIDA_70 : electrical;     constant t70 : real := 70.0;
```

```
begin

      -- A arquitetura é mapeada 9 vezes; cada mapeamento cria uma instância
      --  diferente, cuja interface é conectada independentemente.

      vi_27 : entity work.vi_modelo(vi_comportamental)
            port map (vi_vin => TB_A_INPUT,
                    vi_saida => TB_A_SAIDA_27,
                    vi_t_nr => TB_A_TNR,
                    vi_t_g4 => TB_A_TG4,
                    vi_d_cin => TB_D_CIN,
                    vi_d_cnr => TB_D_CNR,
                    vi_d_cg4 => TB_D_CG4);

      vi_00 : entity work.vi_modelo(vi_comportamental)
            generic map (temp_c => t00)
            port map (vi_vin => TB_A_INPUT,
                    vi_saida => TB_A_SAIDA_00,
                    vi_t_nr => TB_A_TNR,
                    vi_t_g4 => TB_A_TG4,
                    vi_d_cin => TB_D_CIN,
                    vi_d_cnr => TB_D_CNR,
                    vi_d_cg4 => TB_D_CG4);


      vi_10 : entity work.vi_modelo(vi_comportamental)
            generic map (temp_c => t10)
            port map (vi_vin => TB_A_INPUT,
                    vi_saida => TB_A_SAIDA_10,
                    vi_t_nr => TB_A_TNR,
                    vi_t_g4 => TB_A_TG4,
                    vi_d_cin => TB_D_CIN,
                    vi_d_cnr => TB_D_CNR,
                    vi_d_cg4 => TB_D_CG4);


      vi_20: entity work.vi_modelo(vi_comportamental)
            generic map (temp_c => t20)
            port map (vi_vin => TB_A_INPUT,
                    vi_saida => TB_A_SAIDA_20,
                    vi_t_nr => TB_A_TNR,
                    vi_t_g4 => TB_A_TG4,
                    vi_d_cin => TB_D_CIN,
                    vi_d_cnr => TB_D_CNR,
                    vi_d_cg4 => TB_D_CG4);


      vi_30 : entity work.vi_modelo(vi_comportamental)
            generic map (temp_c => t30)
            port map (vi_vin => TB_A_INPUT,
                    vi_saida => TB_A_SAIDA_30,
                    vi_t_nr => TB_A_TNR,
                    vi_t_g4 => TB_A_TG4,
                    vi_d_cin => TB_D_CIN,
                    vi_d_cnr => TB_D_CNR,
                    vi_d_cg4 => TB_D_CG4);


      vi_40 : entity work.vi_modelo(vi_comportamental)
            generic map (temp_c => t40)
```

```
        port map (vi_vin => TB_A_INPUT,
                vi_saida => TB_A_SAIDA_40,
                vi_t_nr => TB_A_TNR,
                vi_t_g4 => TB_A_TG4,
                vi_d_cin => TB_D_CIN,
                vi_d_cnr => TB_D_CNR,
                vi_d_cg4 => TB_D_CG4);


vi_50 : entity work.vi_modelo(vi_comportamental)
        generic map (temp_c => t50)
        port map (vi_vin => TB_A_INPUT,
                vi_saida => TB_A_SAIDA_50,
                vi_t_nr => TB_A_TNR,
                vi_t_g4 => TB_A_TG4,
                vi_d_cin => TB_D_CIN,
                vi_d_cnr => TB_D_CNR,
                vi_d_cg4 => TB_D_CG4);


vi_60 : entity work.vi_modelo(vi_comportamental)
        generic map (temp_c => t60)
        port map (vi_vin => TB_A_INPUT,
                vi_saida => TB_A_SAIDA_60,
                vi_t_nr => TB_A_TNR,
                vi_t_g4 => TB_A_TG4,
                vi_d_cin => TB_D_CIN,
                vi_d_cnr => TB_D_CNR,
                vi_d_cg4 => TB_D_CG4);


vi_70 : entity work.vi_modelo(vi_comportamental)
        generic map (temp_c => t70)
        port map (vi_vin => TB_A_INPUT,
                vi_saida => TB_A_SAIDA_70,
                vi_t_nr => TB_A_TNR,
                vi_t_g4 => TB_A_TG4,
                vi_d_cin => TB_D_CIN,
                vi_d_cnr => TB_D_CNR,
                vi_d_cg4 => TB_D_CG4);

-- Gera estímulo
tb_v_in == 1.5+0.5*sin(2.0*math_pi*freq_v*now);

break on TB_D_CIN, TB_D_CNR, TB_D_CG4;

operacao : process is
begin
        TB_D_CIN <= '1';
        TB_D_CNR <= '1';
        TB_D_CG4 <= '1';
                wait for 20 us;
        TB_D_CIN <= '1';
        TB_D_CNR <= '0';
        TB_D_CG4 <= '0';
                wait for 20 us;
        TB_D_CIN <= '1';
        TB_D_CNR <= '1';
        TB_D_CG4 <= '0';
                wait for 20 us;
        TB_D_CIN <= '0';
```

```
            TB_D_CNR <= '0';
            TB_D_CG4 <= '0';
                  wait for 20 us;
            TB_D_CIN <= '1';
            TB_D_CNR <= '1';
            TB_D_CG4 <= '1';

            wait;
      end process operacao;

end architecture vi_comportamental_tb;
--------------------------------------------------------------------------
```

E.1.3 – Modelo estrutural

E.1.3.1 – Núcleo de conversão

```
entity vi_nuc is
      port (terminal vi_nuc_input : electrical;
      --    quantity vi_nuc_i_in : in real;     -- entrada de 1.84uA (retirada)
            quantity vi_nuc_i_out : out real);
end entity vi_nuc;

----------------------------------------------------------------------------
-- vi_nuc.vhms
-- descricao comportamental do núcleo do conversor tensão-corrente
----------------------------------------------------------------------------
-- Etiqueta_IP_soft % Fornecedor LDCI-UnB % Produto conversor_vi % Versao 1.1
»     % Modelo estrutural % Sub-bloco nucleo
----------------------------------------------------------------------------
architecture vi_nuc of vi_nuc is

      quantity vi_nuc_vin across vi_nuc_iin through vi_nuc_input;

      constant nuc_r_in : resistance :=1.0e+6;
      constant nuc_r_out : resistance :=1.0e+6;

      quantity nuc_out : real;

begin
      vi_nuc_vin == vi_nuc_iin*nuc_r_in;

      nuc_out == ((vi_nuc_vin)-1.5)*10.0e-06;

--    nuc_out == ((vi_nuc_vin)-1.0)*10.0e-06 + 2.5e-06;
--    -- comportamento original do núcleo

      vi_nuc_i_out == nuc_out;

end architecture vi_nuc;
--------------------------------------------------------------------------
```

E.1.3.2 – Referência de corrente

```
entity vi_ref is
      generic (vi_fc_i1 : real := 1.84e-06;
               vi_fc_i2 : real := -7.5e-06);
      port (terminal vi_fc_t1 : electrical;
            terminal vi_fc_t2 : electrical);
end entity vi_ref;

----------------------------------------------------------------------------
-- Este bloco foi testado individualmente, mas não foi incluído no modelo
-- estrutural completo
----------------------------------------------------------------------------
-- Etiqueta_IP_soft % Fornecedor LDCI-UnB % Produto conversor_vi % Versao 1.1
»     % Modelo estrutural % Sub-bloco ref
----------------------------------------------------------------------------
architecture vi_ref of vi_ref is

      quantity vi_fc_out1 through vi_fc_t1;
      quantity vi_fc_out2 through vi_fc_t2;

begin
      vi_fc_out1 == vi_fc_i1;
      vi_fc_out2 == vi_fc_i2;

end architecture vi_ref;
--------------------------------------------------------------------------
```

E.1.3.3 – Estágio intermediário de ganho (G4)

```
entity vi_g4 is
      port (quantity vi_g4_in : in real;
            quantity vi_g4_out : out real);
end entity vi_g4;
----------------------------------------------------------------------------
-- descricao comportamental do estagio de ganho 4 do conversor tensao-corrente
----------------------------------------------------------------------------
-- Etiqueta_IP_soft % Fornecedor LDCI-UnB % Produto conversor_vi % Versao 1.1
»     % Modelo estrutural % Sub-bloco G4
----------------------------------------------------------------------------
architecture vi_g4 of vi_g4 is

      constant ganho : real :=4.0;

begin
      vi_g4_out == ganho*vi_g4_in;
end architecture vi_g4;
--------------------------------------------------------------------------
```

E.1.3.4 – Estágio de saída (G5)

```
entity vi_g5 is
      port (quantity vi_g5_in : in real;
            terminal vi_g5_out : electrical);
```

```
end entity vi_g5;
--------------------------------------------------------------------------
-- descricao comportamental do estagio de saida do conversor tensao-corrente
--------------------------------------------------------------------------
-- Etiqueta_IP_soft % Fornecedor LDCI-UnB % Produto conversor_vi % Versao 1.1
»      % Modelo estrutural % Sub-bloco G5
--------------------------------------------------------------------------
architecture vi_g5 of vi_g5 is

      constant ganho : real :=5.0;
      quantity vi_g5_i_out through vi_g5_out;

begin
      vi_g5_i_out == ganho*vi_g5_in;

end architecture vi_g5;
------------------------------------------------------------------------
```

E.1.3.5 – Superbloco

```
entity vi_macro_r is
end entity vi_macro_r;
--------------------------------------------------------------------------
-- Etiqueta_IP_soft % Fornecedor LDCI-UnB % Produto conversor_vi % Versao 1.1
»      % Modelo estrutural % Sub-bloco superbloco
--------------------------------------------------------------------------
architecture vi_macro_r of vi_macro_r is

      constant freq_v : real :=22.0e3;

      terminal vi_v_in : electrical;
            quantity vi_v_in_v across vi_i_in_v through vi_v_in;
      terminal vi_i_out : electrical;
            quantity vi_i_out_v across vi_i_out_i through vi_i_out;

      quantity vi_int_g4 : real;    -- entrada de g4
      quantity vi_int_g4_g5 : real; -- entre g4 e g5


begin

      vi_v_in_v == 1.5+0.5*sin(2.0*math_pi*freq_v*now);

      vi_i_out_v == vi_i_out_i*1.0e+3;

      vi_nuc : entity work.vi_nuc(vi_nuc)
            port map (vi_nuc_input => vi_v_in,
                    vi_nuc_i_out => vi_int_g4);


      vi_g4 : entity work.vi_g4(vi_g4)
            port map (vi_g4_in => vi_int_g4,
                    vi_g4_out => vi_int_g4_g5);


      vi_g5 : entity work.vi_g5(vi_g5)
            port map (vi_g5_in => vi_int_g4_g5,
                    vi_g5_out => vi_i_out);
```

```
	operacao_macro : process is
	begin
		wait;
	end process operacao_macro;

end architecture vi_macro_r;
-------------------------------------------------------------------------
```

E.1.3.6 – Testbench de *vi_nuc*

Para o teste do núcleo (assim como em algumas outras plataformas de teste), o *testbench* não gera o sinal, mas instancia um bloco externo, *sinal_v*, que fornece sinal de tensão senoidal.

```
architecture vi_nuc_tb of vi_nuc_tb is
	terminal TB_A_SINAL : electrical;
	terminal TB_NUC_OUT : electrical;
		quantity TB_NUC_I_OUT through TB_NUC_OUT;
--	quantity TB_I_REF : current;


begin

	vi_nuc : entity work.vi_nuc(vi_nuc)
		port map (vi_nuc_input => TB_A_SIGNAL,
--				vi_nuc_i_in => TB_I_REF,
				vi_nuc_i_out => TB_NUC_I_OUT);

	stim : entity work.s_sinal(s_sinal)
		port map (EXT_V => TB_A_SINAL);

--	TB_I_REF == 1.84e-06;

	operacao : process is
	begin
		REPORT "Testbench iniciado";
		wait;
	end process operacao;

end architecture vi_nuc_tb;
-------------------------------------------------------------------------
```

E.1.3.7 – Testbench de *vi_ref*

```
entity vi_fc_tb is
	generic (tb_fc_gout1 : real;
	         tb_fc_gout2 : real);
end entity vi_fc_tb;

architecture vi_fc_tb of vi_fc_tb is
	terminal TB_REF_OUT1, TB_REF_OUT2 : electrical;
		quantity tb_ref_out_v1 across tb_ref_out_i1 through TB_REF_OUT1;
		quantity tb_ref_out_v2 across tb_ref_out_i2 through TB_REF_OUT2;


begin

	vi_ref : entity work.vi_ref(vi_ref)
	port map (vi_fc_t1 => TB_FC_OUT1,
	          vi_fc_t2 => TB_FC_OUT2);
```

```
        tb_fc_out_v1 == tb_fc_out_i1*(1.0e+04);
        tb_fc_out_v2 == tb_fc_out_i2*(1.0e+04);

        operacao : process is
        begin
              wait;
        end process operacao;

end architecture vi_fc_tb;
-------------------------------------------------------------------------
```

E.1.3.8 – Testbench de *vi_g4*

```
architecture vi_g4_tb of vi_g4_tb is
        constant freq_v : real :=22.0e3;

        terminal TB_G4_IN : electrical;
              quantity tb_g4_i_in through TB_G4_IN;
        quantity TB_G4_OUT : current;

begin

        vi_g4 : entity work.vi_g4(vi_g4)
              port map (vi_g4_in => TB_G4_I_IN,
                        vi_g4_out => TB_G4_OUT);


        tb_g4_in == 5.0e-06*sin(2.0*math_pi*freq_v*now);

        operacao : process is
        begin
              wait;
        end process operacao;

end architecture vi_g4_tb;
-------------------------------------------------------------------------
```

E.1.3.9 – Testbench de *vi_g5*

```
architecture vi_g5_tb of vi_g5_tb is
        constant freq_v : real :=25.0e3;

        quantity TB_G5_IN : current;
        terminal TB_G5_OUT : electrical;

begin

        vi_g5 : entity work.vi_g5(vi_g5)
              port map (vi_g5_in => TB_G5_IN,
                        vi_g5_out => TB_G5_OUT);

        TB_G5_IN == 20.0e-06*sin(2.0*math_pi*freq_v*now);

        operacao : process is
        begin
              wait;
        end process operacao;

end architecture vi_g5_tb;
```

E.2 – MODELAGEM DO CONVERSOR A/D

E.2.1 – Modelo funcional

```
entity ad_modelo is
      generic (ad_offset_b : real := 0.0;
               ad_gerr_b : real := 0.0;
               ad_INL : integer := 0;
                  ad_INL_vlow : real := 3.1250e-06;
                  ad_INL_vhigh : real := 4.6875e-06;
               ad_DNL_in : bit_vector := ('0','0','0','0','0','0','0','0');
               ad_DNL_out : bit_vector := ('0','0','0','0','0','0','0','0') );

      port (terminal AD_A_INPUT : electrical;
            signal AD_D_SAIDA : out bit;
            signal AD_D_CLK : in bit;
            signal AD_D_ON_OFF : in bit;
            signal ad_d_report_out : out bit :='0';
            signal ad_d_eoc : out bit := '0');
end entity ad_modelo;

---------------------------------------------------------------------------
-- A interface do modelo funcional e do modelo comportamental são equivalen-
-- tes, portanto só a arquitetura muda. Com todas as portas genéricas tendo
-- valor inicial atribuído - e nulo -, não é necessário utilizá-las aqui.
---------------------------------------------------------------------------
-- Etiqueta_IP_soft % Fornecedor LDCI-UnB % Produto conversor_ad % Versao 1.0
»     % Modelo funcional
---------------------------------------------------------------------------
architecture ad_funcional of ad_modelo is
      constant ref_zero : real :=0.0;
      constant ref_100 : real :=100.0E-6;
      constant ad_r_in : resistance :=2400.0;

      -- Tensão e corrente na entrada analógica
      quantity ad_v_in across ad_i_in through AD_A_INPUT;

      -- Declaração do subtipo que vai ser usado para armazenar os bits
      subtype out_byte is bit_vector(7 downto 0);


begin

      ad_v_in == ad_r_in*ad_i_in;

      ad_ideal_conversao: process is
            -- Declara uma variável como um vetor de 8 bits
            variable output_byte : out_byte;

            -- Declara variáveis a serem usadas na conversao
            variable i_smp : real;
            variable i_smp_reg : real;

            variable loop_count : integer := 1;

      begin

            --  Os procedimentos abaixo, no loop 'amostragem_e_conversão', são
            --    repetidos continuamente enquanto o A/D permanecer ligado.
```

```
            amostragem_e_conversao : WHILE AD_D_ON_OFF = '1' LOOP

            exit amostragem_e_conversao when AD_D_ON_OFF='0';
--          -- A linha acima deveria interromper a operação do conversor
--          -- a qualquer instante, se o sinal on/off for zerado.
--          -- Mas sua implementação não funcionou nas ferramentas de simulação
--          -- utilizadas.


                  wait until AD_D_CLK = '1';
                  -- Começa a conversão no relógio seguinte

                  i_smp := -ad_i_in;            -- Amostra a corrente de entrada
                  i_smp_reg := i_smp;           -- Registra o valor

                  -- Atribui '0' ao bit que indica conversao
                  ad_report_out <= '0';

                  -- Zera a saída, caso estivesse com valor armazenado
                  ad_d_saida <= '0';

                  -- Indica o fim de uma conversão
                  IF loop_count /= 1 THEN
                        ad_d_eoc <= '1';
                  END IF;

            -- Confere se a entrada está dentro da faixa especificada;
            -- se não estiver, aguarda o próximo ciclo de amostragem:
            check_range : IF i_smp < (-ref_100) THEN
                  report "Entrada fora da faixa especificada" severity warning;
                  ad_d_saida <= '0';
                  wait for 20 us;

            ELSIF i_smp > ref_100 THEN
                  report "Entrada fora da faixa especificada" severity warning;
                  ad_d_saida <= '1';
                  wait for 20 us;

            ELSE
                  report "Entrada de " & real'image(i_smp) & " A" severity note;

---------------------------------------------------------------
-- Conversão de 8 bits
---------------------------------------------------------------
                  conversao : FOR i IN 7 DOWNTO 0 LOOP
                        IF i_smp >= ref_zero THEN
                              output_byte(i) := '1';
                              i_smp := i_smp*2.0 - ref_100;

                        ELSE
                              output_byte(i) := '0';
                              i_smp := i_smp*2.0 + ref_100;

                        END IF;
                  END LOOP conversao;
---------------------------------------------------------------
                  wait until AD_D_CLK = '1';
                        -- zera a indicação de conversão anterior
                        ad_d_eoc <= '0';

                  -- Aguarda para que o ciclo de conversao completo dure 20us
```

```
                WAIT FOR 19.4375 us;

                -- Corta o ciclo caso o conversor tenha sido desligado durante
                -- a espera (desnecessário se o comando 'exit when' funcionar)
                      check_reset : IF AD_D_ON_OFF = '0' THEN
                            EXIT amostragem_e_conversao;
                      END IF check_reset;
--------------------------------------------------------------
-- Saída
--------------------------------------------------------------

                ad_report_out <= '1';   -- Indica que os bits em ad_d_saida
                                        -- são o resultado da conversão

                saida : FOR i IN 7 DOWNTO 0 LOOP
                      IF AD_D_ON_OFF = '0' THEN
                            EXIT amostragem_e_conversao;
                      ELSE
                      -- Um bit a cada ciclo de relógio
                            wait until AD_D_CLK = '1';
                            ad_d_saida <= output_byte(i);
                      END IF;
                END LOOP saida;
--------------------------------------------------------------

          END IF check_range;

                loop_count := loop_count + 1;

          END LOOP amostragem_e_conversao;

-- Caso o A/D tenha sido desligado, garante condições iniciais e espera até que
-- seja ligado novamente.
          report "Conversor A/D desligado";
          loop_count := 1;
          ad_d_eoc <= '0'; ad_report_out <= '0'; ad_d_saida <= '0';
          wait on AD_D_ON_OFF;

      end process ad_ideal_conversao;
end architecture ad_funcional;
--------------------------------------------------------------
```

E.2.2 – Modelo comportamental

```
---------------------------------------------------------------------------
-- Etiqueta_IP_soft % Fornecedor LDCI-UnB % Produto conversor_ad % Versao 1.0
»     % Modelo comportamental
---------------------------------------------------------------------------
architecture ad_comportamental of ad_comportamental is
      constant ad_res : integer := 8;
      constant ref_zero : real :=0.0;
      constant ref_100 : real :=100.0E-6;
      constant ad_r_in : resistance :=2400.0;
      quantity ad_v_in across ad_i_in through AD_A_INPUT;

      subtype out_byte is bit_vector(7 downto 0);

begin
```

```
    ad_v_in == ad_r_in*ad_i_in;

    ad_comp_operacao: process is
          variable output_byte : out_byte;
          variable i_smp : real;
          variable i_smp_reg : real;

          variable loop_count : integer := 1;

    -- Cria variáveis a partir dos parâmetros, para utilizar na modelagem
          variable ad_offset : real := ad_offset_b*0.78125e-06;
          variable ad_gerr : real := (128.0 - ad_gerr_b)/128.0;
                -- pelo fato de a curva característica ser simétrica,
                --  usa-se 128 (resulta em erro de 1 bit no fim da escala)

          begin
          amostragem_e_conversao : WHILE AD_D_ON_OFF = '1' LOOP
          exit amostragem_e_conversao when AD_D_ON_OFF='0';

                wait until AD_D_CLK = '1';
----------------------------------------------------------------------------
-- AMOSTRAGEM - considera offset, erro de ganho e INL
----------------------------------------------------------------------------
                i_smp := -(ad_i_in + ad_offset)*(ad_gerr);
                i_smp_reg := i_smp;

          INL : IF ad_INL /= 0 and (i_smp_reg > ad_INL_vlow)
»               and (i_smp_reg < ad_INL_vhigh) THEN
                      i_smp := ad_INL_vlow + real(ad_INL)*0.78125e-06;
          END IF INL;

                ad_report_out <= '0';
                ad_d_saida <= '0';

                IF loop_count /= 1 THEN
                      ad_d_eoc <= '1';
                END IF;
----------------------------------------------------------------------------
-- Conversão de 8 bits
----------------------------------------------------------------------------
                conversao : FOR i IN (ad_res -1)  DOWNTO 0 LOOP
                      IF i_smp >= (ref_zero) THEN
                            output_byte(i)  := '1';
                            i_smp := i_smp*2.0 - ref_100;

                      ELSE
                            output_byte(i)  := '0';
                            i_smp := i_smp*2.0 + ref_100;

                      END IF;

                END LOOP conversao;

          -- DNL:
                DNL : IF output_byte = ad_DNL_in THEN
                      output_byte := ad_DNL_out;
                END IF DNL;
----------------------------------------------------------------------------
                wait until AD_D_CLK = '1';
                      ad_d_eoc <= '0';
```

```
                  WAIT FOR 19.4375 us;

                  check_reset : IF AD_D_ON_OFF = '0' THEN
                              EXIT amostragem_e_conversao;
                  END IF check_reset;

                  ad_report_out <= '1';
----------------------------------------------------------------------------
-- Saída
----------------------------------------------------------------------------
                  saida : FOR i IN 7 DOWNTO 0 LOOP
                        IF AD_D_ON_OFF = '0' THEN
                              EXIT amostragem_e_conversao;
                        ELSE
                              wait until AD_D_CLK = '1';
                              ad_d_saida <= output_byte(i);
                        END IF;
                  END LOOP saida;
----------------------------------------------------------------------------
                  loop_count := loop_count + 1;

            END LOOP amostragem_e_conversao;

            report "Conversor A/D desligado";
            loop_count := 1;
            ad_d_eoc <= '0'; ad_report_out <= '0'; ad_d_saida <= '0';
            wait on AD_D_ON_OFF;

      end process ad_comp_operacao;
end architecture ad_comportamental;
----------------------------------------------------------------------------
```

E.2.3 – Modelo estrutural

E.2.3.1 – Sub-bloco *S/H*

```
entity ad_sh is
      generic ( ad_offset_b : real := 0.0;
              ad_gerr_b : real := 0.0;
              ad_INL : integer := 0;
                  ad_INL_vlow : real := 3.1250e-06;
                  ad_INL_vhigh : real := 4.6875e-06);
      port (terminal  ad_sh_entrada : electrical;
            terminal ad_sh_t_saida  : electrical;
            signal ad_sh_on_off : in bit;
            signal ad_sh_d_eoc : in bit;
            signal ad_sh_clk : in bit);
end entity ad_sh;
----------------------------------------------------------------------------
-- Etiqueta_IP_soft % Fornecedor LDCI-UnB % Produto conversor_ad % Versao 1.0
»     % Modelo estrutural % Sub-bloco S/H
----------------------------------------------------------------------------
architecture ad_sh of ad_sh is
      constant ad_res : integer := 8;
      constant ad_r_in : resistance :=2400.0;
      constant ad_r_out : resistance :=1.0e06;

      signal smp_bit : bit;
```

```
      -- Declaração de tensões e correntes:
      quantity ad_sh_vin across ad_sh_iin through ad_sh_entrada;
      quantity ad_sh_saida through ad_sh_t_saida;

      shared variable i_smp : real;

begin

      ad_sh_vin == ad_r_in*ad_sh_iin;

      ad_sh_saida == i_smp;

      ad_sample: process is

            -- Considera os efeitos não ideais de offset e erro de ganho:
            variable ad_offset : real := ad_offset_b*0.78125e-06;
            variable ad_gerr : real := (128.0 - ad_gerr_b)/128.0;

            begin
            IF ad_sh_on_off = '0' THEN
                  i_smp := 0.0;
                  wait on ad_sh_on_off;
            END IF;

            amostragem : WHILE ad_sh_on_off = '1' LOOP
                  wait until ad_sh_clk = '1';
---------------------------------------------------------------
-- AMOSTRAGEM
---------------------------------------------------------------
                  i_smp := -(ad_sh_iin + ad_offset)*(ad_gerr);

            -- Considera INL:
            INL : IF ad_INL /= 0 and (i_smp > ad_INL_vlow)
»                 and (i_smp < ad_INL_vhigh) THEN
                        i_smp := ad_INL_vlow + real(ad_INL)*0.78125e-06;
            END IF INL;

---------------------------------------------------------------
            -- Mantém o sinal durante todo o ciclo:
                  wait until ad_sh_on_off = '0' or ad_sh_d_eoc = '1';

            END LOOP amostragem;

      end process ad_sample;

      break on smp_bit, ad_sh_on_off, ad_sh_d_eoc, ad_sh_clk;

end architecture ad_sh;
----------------------------------------------------------------------------
```

E.2.3.2 – Sub-bloco *Memo*

```
entity ad_memo is

      port (terminal ad_memo_entrada : electrical;
            terminal ad_memo_saida : electrical;
            signal ad_memo_lc : out bit;
            signal ad_memo_clk : in bit;
            signal ad_memo_on_off : in bit;
```

```
          signal ad_memo_d_eoc : in bit := '0');
end entity ad_memo;

----------------------------------------------------------------------------
-- Etiqueta_IP_soft % Fornecedor LDCI-UnB % Produto conversor_ad % Versao 1.0
»     % Modelo estrutural % Sub-bloco Memo
----------------------------------------------------------------------------
architecture ad_memo of ad_memo is
      constant ad_res : integer := 8;
      constant ref_zero : real :=0.0;
      constant ref_100 : real :=100.0E-6;

      constant ad_r_in : resistance :=2400.0;
      constant ad_r_out : resistance :=1.0e06;

      quantity ad_memo_vin across ad_memo_iin through ad_memo_entrada;
      quantity ad_memo_vout across ad_memo_iout through ad_memo_saida;

      shared variable ad_i_ciclos : real;

begin

      ad_memo_vin == ad_r_in*ad_memo_iin;

      ad_memo_iout == ad_i_ciclos;

      ad_memo_est: process is
            variable i_memo_smp : real;

            begin
            ad_memo_lc <= '0';

            IF ad_memo_on_off = '0' THEN
                  ad_i_ciclos := 0.0;
                  wait on ad_memo_on_off;
            END IF;
----------------------------------------------------------------------------
            ad_memo : WHILE ad_memo_on_off = '1' LOOP
                  wait until ad_memo_clk = '1';
                  wait until ad_memo_clk = '1'; -- Espera amostragem
----------------------------------------------------------------------------
                  ciclos_memo : FOR i IN 0 TO (ad_res-1) LOOP
                        i_memo_smp := - ad_memo_iin;
                        ad_i_ciclos := i_memo_smp*(2.0);
                        ad_memo_lc <= '1';

                        wait until ad_memo_clk = '1'; -- espera comparação
                        wait until ad_memo_clk = '1'; -- reinicia loop
                        ad_memo_lc <= '1';
                  END LOOP ciclos_memo;
----------------------------------------------------------------------------
                  ad_i_ciclos := 0.0;
                  wait until ad_memo_on_off = '0' or ad_memo_d_eoc = '1';
                  ad_memo_lc <= '0';

            END LOOP ad_memo;

            ad_memo_lc <= '0';

      end process ad_memo_est;
```

```
      break on ad_memo_on_off, ad_memo_d_eoc, ad_memo_clk;

-- O trecho abaixo seria usado para descrever o funcionamento do bloco
-- utilizando PROCEDURALS, estruturas em VHDL-AMS que permitem controlar o
-- funcionamento analógico de uma entidade com declarações seqüenciais.
-- Entretanto, as ferramentas utilizadas ainda não suportam o uso de
-- procedurals.

--    ad_memo_fp : procedural is
--          variable i_memo_smp_p : real;
--          variable loop_count : bit;
--
--          begin
--          IF ad_memo_on_off = '0' THEN
--                ad_memo_iout := 0.0;
--                loop_count <= '1';
--          ELSE
---------------------------------------------------------------------
--                c_memo : FOR i IN 0 TO (ad_res-1) LOOP
--                      IF loop_count = '1' THEN
--                            i_memo_smp_p := - ad_memo_iin;
--                            ad_memo_iout := i_memo_smp_p;
--                            loop_count <= '0';
--                            wait until ad_memo_clk = '1'; -- espera comparação
--                            wait until ad_memo_clk = '1'; -- reinicia loop
--                      ELSE
--                            i_memo_smp_p := - ad_memo_iin_c;
--                            ad_memo_iout := i_memo_smp_p*(2.0);
--                            loop_count <= '0';
--                            wait until ad_memo_clk = '1'; -- espera comparação
--                            wait until ad_memo_clk = '1'; -- reinicia loop
--                      END IF;
--                            loop_count <= '0';
--                END LOOP c_memo;
-----------------------------------------------------------------------
--          END IF;
--    END PROCEDURAL ad_memo_fp;

end architecture ad_memo;
-----------------------------------------------------------------------
```

E.2.3.3 – Sub-bloco *Ref*

```
entity ad_ref is
      port (terminal ad_ref_saida : electrical;
            signal ad_ref_clk : in bit;
            signal ad_ref_comp : in bit;
            signal ad_ref_on_off : in bit;
            signal ad_ref_d_eoc : in bit := '0');
end entity ad_ref;
---------------------------------------------------------------------------
-- Etiqueta_IP_soft % Fornecedor LDCI-UnB % Produto conversor_ad % Versao 1.0
»     % Modelo estrutural % Sub-bloco Ref
---------------------------------------------------------------------------
architecture ad_ref of ad_ref is
      constant ad_res : integer := 8;
      constant ref_zero : real :=0.0;
      constant ref_100 : real :=100.0E-6;

      constant ad_r_out : resistance :=1.0e06;
```

```
	quantity ad_ref_vout across ad_ref_iout through ad_ref_saida;

	-- Subtipo do qual objetos aceitam valores -1, 0 ou 1
	subtype ref_var is integer range -1 to 1 ;

	shared variable ad_i_var : ref_var := 0;

begin

	ad_ref_iout == real(ad_i_var)*(-1.0)*100.0e-06;

	ad_ref_est: process is
		variable comparacao_anterior : bit;

	begin

		IF ad_ref_on_off = '0' THEN
			ad_i_var := 0;
		END IF;


		WHILE ad_ref_on_off = '1' LOOP
			comparacao_anterior := '0';
			ad_i_var := 0;
			wait until ad_ref_clk = '1';
			wait until ad_ref_clk = '1';  -- espera amostragem (em ad_sh)
			wait until ad_ref_clk = '1';  -- espera amostragem (em ad_memo)

		FOR i IN 1 TO (ad_res) LOOP

			wait until ad_ref_clk = '0';  -- para que a corrente não mude
			                              -- no instante da comparação
			comparacao_anterior := ad_ref_comp;
			                        -- atualiza ref antes de atualizar memo

	-- Saída assume valor dependente do resultado da comparação anterior
			CASE comparacao_anterior IS
				when '0' =>
					ad_i_var := -1;
				when '1' =>
					ad_i_var := 1;
			END CASE;
			comparacao_anterior := ad_ref_comp;

			wait until ad_ref_clk = '0';

		END LOOP; -- for
		ad_i_var := 0;

		END LOOP; -- while
		wait on ad_ref_d_eoc, ad_ref_on_off;

	end process ad_ref_est;
	break on ad_ref_comp, ad_ref_clk;

end architecture ad_ref;
----------------------------------------------------------------------------
```

E.2.3.4 – Sub-bloco *Comp*

```
entity ad_comparador is
      port (terminal ad_comp_entrada : electrical;
            signal ad_comp_clk : in bit;
            signal ad_comp_on_off : in bit;
            signal ad_comp_d_eoc : in bit := '0';
            signal ad_comp_saida : out bit := '0';
            signal ad_comp_report_out : out bit := '0');
end entity ad_comparador;

----------------------------------------------------------------------------
-- Etiqueta_IP_soft % Fornecedor LDCI-UnB % Produto conversor_ad % Versao 1.0
»     % Modelo estrutural % Sub-bloco Comp
----------------------------------------------------------------------------

architecture ad_comparador of ad_comparador is
      constant ad_res : integer := 8;
      constant ref_zero : real :=0.0;
      constant ref_100 : real :=100.0E-6;
      constant ad_r_in : resistance :=2400.0;

      quantity ad_comp_vin across ad_comp_iin through ad_comp_entrada;

begin

      ad_comp_vin == ad_r_in*ad_comp_iin;

      ad_comp_est: process is
            variable i_comp_smp : real;
            variable i_comp_smp_reg : real;

      begin
            IF ad_comp_on_off = '0' THEN
                  ad_comp_saida <= '0';
                  wait on ad_comp_on_off;
            END IF;

            ad_comparacao : WHILE ad_comp_on_off = '1' LOOP

                  wait until ad_comp_clk = '1'; -- espera amostragem em ad_sh
                  wait until ad_comp_clk = '1'; -- espera atualizar memo
                  wait until ad_comp_clk = '1'; -- compara:
---------------------------------------------------------------
-- Comparação
                  i_comp_smp_reg := ad_comp_iin;      -- registra o valor

                  ad_comp_report_out <= '1';

                  comparacao : FOR i IN 1 TO ad_res LOOP
                        i_comp_smp := ad_comp_iin;

                        IF i_comp_smp >= 0.0 THEN
                              ad_comp_saida <= '1';
                        ELSE
                              ad_comp_saida <= '0';
                        END IF;

                        wait until ad_comp_clk = '1';
                                          -- corrente amostrada em memo
                        wait until ad_comp_clk = '1';
```

```
                                        -- reinicia o loop
                        END LOOP comparacao;

                        ad_comp_report_out <= '0';
                        wait until ad_comp_on_off = '0' or ad_comp_d_eoc = '1';

                END LOOP ad_comparacao;
end process ad_comp_est;

end architecture ad_comparador;
----------------------------------------------------------------------------
```

E.2.3.5 – Sub-bloco *Saída*

```
entity ad_saida is
      generic ( ad_DNL_in : bit_vector := ('0','0','0','0','0','0','0','0');
                ad_DNL_out : bit_vector := ('0','0','0','0','0','0','0','0') );

      port (signal ad_saida_in : in bit;
            signal ad_saida_clk : in bit;
            signal ad_saida_on_off : in bit;
            signal ad_saida_report_in : in bit;
            signal ad_saida_out_bit : out bit;
            signal ad_saida_d_eoc : out bit := '0';
            signal ad_saida_report_out : out bit := '0');
end entity ad_saida;
----------------------------------------------------------------------------
-- Etiqueta_IP_soft % Fornecedor LDCI-UnB % Produto conversor_ad % Versao 1.0
»     % Modelo estrutural % Sub-bloco Saida
----------------------------------------------------------------------------
architecture ad_saida of ad_saida is
      constant ad_res : integer := 8;
      constant t_clk : time := 62.5 ns;

      subtype out_byte is bit_vector(7 downto 0);

begin
      ad_saida_est: process is
            variable output_byte : out_byte;
            variable loop_count : integer := 1;

      begin
            IF ad_saida_on_off = '0' THEN
                  ad_saida_d_eoc <= '0';
                  ad_saida_out_bit <= '0';
                  wait on ad_saida_on_off;
            END IF;

            ad_saida_est : WHILE ad_saida_on_off = '1' LOOP
----------------------------------------------------------------------------
                  IF loop_count /= 1 THEN
                        ad_saida_d_eoc <= '1';
                  END IF;

                  IF ad_saida_report_in /= '1' THEN
                        -- Espera chegarem os bits do comparador
                        wait until ad_saida_report_in = '1';
                        ad_saida_d_eoc <= '0';
                  END IF;
```

```
            armazena : FOR i IN (ad_res -1) DOWNTO 0 LOOP
                  wait until ad_saida_clk = '1';
                                    -- memo e ref se atualizam
                  output_byte(i) := ad_saida_in;
                  wait until ad_saida_clk = '1';
                                    -- comparador se atualiza
                  ad_saida_d_eoc <= '0';
            END LOOP armazena;

      -- Aguarda antes de entregar os bits na saída
            wait for real((320 - (3*ad_res - 4)))*t_clk;
                  -- 1 para iniciar; 1 para amostrar em ad_sh
                  -- 2*ad_res pra gerar os bits da conversao
                  -- 1 para armazenar o LSB na saída
                  -- 1 ad_saida aguarda antes de sair do loop
                  -- 1*ad_res para o loop de saída
            ad_saida_report_out <= '1';

         -- Loop de saída:
            saida: FOR i IN (ad_res -1) DOWNTO 0 LOOP
                  ad_saida_out_bit <= output_byte(i);
                  wait until ad_saida_clk = '1';
            END LOOP saida;

            ad_saida_report_out <= '0';
            loop_count := loop_count + 1;

         END LOOP ad_saida_est;

      end process ad_saida_est;
end architecture ad_saida;
---------------------------------------------------------------------------
```

E.2.3.6 – Sub-bloco *ad_sum*

O código do sub-bloco *ad_sum*, que não faz parte da estrutura do A/D mas foi incluído para corrigir erros na simulação do superbloco, é apresentado abaixo.

```
entity ad_sum is
      port (terminal ad_sum_entrada_sh : electrical;
            terminal ad_sum_entrada_c : electrical;
            terminal ad_sum_saida : electrical;
            signal ad_sum_lc : in bit);
end entity ad_sum;
---------------------------------------------------------------------------
architecture ad_sum of ad_sum is
      constant ad_res : integer := 8;
      constant ref_zero : real :=0.0;
      constant ref_100 : real :=100.0E-6;

      constant ad_r_in : resistance :=2400.0;
      constant ad_r_out : resistance :=1.0e06;

      quantity ad_sum_vin_sh across ad_sum_iin_sh through ad_sum_entrada_sh;
      quantity ad_sum_vin_c across ad_sum_iin_c through ad_sum_entrada_c;
      quantity ad_sum_vout across ad_sum_iout through ad_sum_saida;

begin
```

```
	ad_sum_vin_sh == ad_r_in*ad_sum_iin_sh;
	ad_sum_vin_c == ad_r_in*ad_sum_iin_c;

	IF ad_sum_lc = '0' USE
		ad_sum_iout == ad_sum_iin_sh/2.0;
		-- Para manter o sub-bloco MEMO multiplicando por 2,
		-- no primeiro ciclo o sinal entregue é metade da corrente
		-- amostrada em S/H.
	ELSE
		ad_sum_iout == ad_sum_iin_c;
	END USE;

	break on ad_sum_lc;

end architecture ad_sum;
---------------------------------------------------------------------------
```

E.2.3.7 – Superbloco

```
entity ad_macro is
end entity ad_macro;
---------------------------------------------------------------------------
architecture ad_macro of ad_macro is
	constant t_clk : time := 62.5 ns;
	constant freq_i : real := 22.0e+3;
	constant ad_r_in : resistance :=2400.0;
	constant ad_r_out : resistance :=1.0e06;

	signal TB_D_CLK, TB_AD_ON_OFF, TB_D_EOC, TB_COMP_SAIDA, TB_MEMO_LC,
»		TB_REPORT_COMP, TB_REPORT_SAIDA, TB_SAIDA_IN, TB_AD_OUTPUT : bit;


	terminal TB_AD_INPUT, TB_INTERFACE_COMP, TB_INTERFACE_SH,
»		TB_INTERFACE_SUM : electrical;
		quantity ad_v_in across ad_i_in through TB_AD_INPUT;

begin

	-- Sinal de corrente na faixa de entrada do A/D:
	ad_i_in == 100.0e-06*sin(2.0*math_pi*freq_i*now);

	ad_sh : entity work.ad_sh(ad_sh)
		port map (ad_sh_entrada => TB_AD_INPUT,
			ad_sh_t_saida => TB_INTERFACE_SH,
			ad_sh_on_off => TB_AD_ON_OFF,
			ad_sh_d_eoc => TB_D_EOC,
			ad_sh_clk => TB_D_CLK);

	ad_sum : entity work.ad_sum(ad_sum)
		port map (ad_sum_entrada_sh => TB_INTERFACE_SH,
			ad_sum_entrada_c => TB_INTERFACE_COMP,
			ad_sum_saida => TB_INTERFACE_SUM,
			ad_sum_lc => TB_MEMO_LC);


	ad_memo : entity work.ad_memo(ad_memo)
		port map (ad_memo_entrada => TB_INTERFACE_SUM,
			ad_memo_saida => TB_INTERFACE_COMP,
```

```vhdl
                        ad_memo_on_off => TB_AD_ON_OFF,
                        ad_memo_d_eoc => TB_D_EOC,
                        ad_memo_clk => TB_D_CLK,
                        ad_memo_lc => TB_MEMO_LC);

        ad_ref : entity work.ad_ref(ad_ref)
                port map (ad_ref_clk => TB_D_CLK,
                        ad_ref_comp => TB_COMP_SAIDA,
                        ad_ref_on_off => TB_AD_ON_OFF,
                        ad_ref_d_eoc => TB_D_EOC,
                        ad_ref_saida => TB_INTERFACE_COMP);

        ad_comparador : entity work.ad_comparador(ad_comparador)
                port map (ad_comp_entrada => TB_INTERFACE_COMP,
                        ad_comp_clk => TB_D_CLK,
                        ad_comp_on_off => TB_AD_ON_OFF,
                        ad_comp_d_eoc => TB_D_EOC,
                        ad_comp_saida => TB_COMP_SAIDA,
                        ad_comp_report_out => TB_REPORT_COMP);

        ad_saida : entity work.ad_saida(ad_saida)
                port map (ad_saida_in => TB_COMP_SAIDA,
                        ad_saida_clk => TB_D_CLK,
                        ad_saida_on_off => TB_AD_ON_OFF,
                        ad_saida_report_in => TB_REPORT_COMP,
                        ad_saida_out_bit => TB_AD_OUTPUT,
                        ad_saida_d_eoc => TB_D_EOC,
                        ad_saida_report_out => TB_REPORT_SAIDA);

        operacao : process is
        begin

                TB_AD_ON_OFF <= '0';

        ciclos_1 : FOR i in 1 TO 4 LOOP
                TB_D_CLK <= '1';
                        wait for t_clk*0.5;
                TB_D_CLK <= '0';
                        wait for t_clk*0.5;
        END LOOP ciclos_1;

        -- Liga o A/D
                TB_D_CLK <= '1';
                TB_AD_ON_OFF <= '1';            -- A/D é iniciado no próximo clk = '1'
                        wait for t_clk*0.5;
                TB_D_CLK <= '0';
                        wait for t_clk*0.5;

        -- Gera ciclos de relógio para observar o funcionamento
                muitos_ciclos_1 : FOR i in 1 TO 600 LOOP
                        TB_D_CLK <= '1';
                                wait for t_clk*0.5;
                        TB_D_CLK <= '0';
                                wait for t_clk*0.5;
                END LOOP muitos_ciclos_1;

                wait;
        end process operacao;
end architecture ad_macro;
```

APÊNDICE F – CÓDIGO MATLAB DA MODELAGEM DO A/D

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% Código para simulação de características não-ideais do conversor A/D
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
clear

%% Definição de algumas constantes:
ref_zero = 0.0;
ref_100 = 100.0e-06;
n=8192;

%% Criação do estímulo:
t=[0:1/(n-1):1];
ad_i_in = (-100.0 + t*200)*1e-06;
ad_i_in_ideal = ad_i_in;


%% Parâmetros de não-idealidade %%
% (Valor "0" nos parâmetros resulta em comportamento ideal)

ad_offset_b = 0;            %
    display (ad_offset_b);

ad_gerr_b = 0;              %
    display (ad_gerr_b)

ad_INL = 0;                 %
    ad_INL_vlow = 3.3e-06
    ad_INL_vhigh = 3.92e-06;

ad_DNL_in  = [0 0 0 0 0 0 0 0];     %
ad_DNL_out = [0 0 0 0 0 0 0 0];     %

%% Ajuste dos valores para a faixa de operação:
ad_offset = ad_offset_b*0.78125e-06;
ad_gerr = (128.0 - ad_gerr_b)/128.0;

%%%%%% Operação %%%%%%

%   Obs.: foi gerada, paralelamente à resposta comportamental, uma resposta
%   ideal, para comparação.

for z=1:n

%% Amostragem

    %% OFFSET e ERRO DE GANHO
    i_smp = (ad_i_in(z) - ad_offset)*(ad_gerr);
        i_smp_ideal = ad_i_in_ideal(z);

    i_smp_reg = i_smp;
        i_smp_reg_ideal = i_smp_ideal;

    i_smp_norm = ((i_smp + 100e-06)/200e-06)*255;
        i_smp_ideal_norm = ((i_smp_ideal + 100e-06)/200e-06)*255;
```

```
    %% INL
    if (ad_INL ~= 0.0) & (i_smp > ad_INL_vlow) & (i_smp < ad_INL_vhigh)
        i_smp = i_smp + ad_INL*0.78125e-06;
    end;



%% Conversão
    for i=8:-1:1                % LSB é a primeira coluna, e o MSB é a ultima
        if i_smp >= 0.0
            output_byte(i) = 1;
            i_smp = i_smp*2.0 - 100.0e-06;
        else
            output_byte(i) = 0;
            i_smp = i_smp*2.0 + 100.0e-06;
        end;

        % ideal:
        if i_smp_ideal >= 0.0
            output_byte_ideal(i) = 1;
            i_smp_ideal = i_smp_ideal*2.0 - 100.0e-06;
        else
            output_byte_ideal(i) = 0;
            i_smp_ideal = i_smp_ideal*2.0 + 100.0e-06;
        end;
        %

    end;


%% Saída
    if output_byte == ad_DNL_in
        output_byte = ad_DNL_out;
    end;

ad_output_byte(z) = binvec2dec(output_byte);
ad_output_byte_ideal(z) = binvec2dec(output_byte_ideal);

end;

%% Geração do gráfico
figure,plot(ad_i_in, ad_output_byte, 'r', ad_i_in_ideal, ad_output_byte_ideal,
'--b','LineWidth',3);
```